Stick: A Harmonious Fusion of Buffer-based and Learning-based Approach for Adaptive Streaming

Tianchi Huang*, Chao Zhou†¶, Rui-Xiao Zhang*, Chenglei Wu*, Xin Yao*, Lifeng Sun‡§¶
*Beijing Key Lab of Networked Multimedia, Department of Computer Science and Technology, Tsinghua University
†Beijing Kuaishou Technology Co., Ltd., China
‡BNRist, Dept. of Computer Science and Technology, Tsinghua University
§Key Laboratory of Pervasive Computing (Tsinghua University), Ministry of Education, China
¶Corresponding Author. {htc19@mails.,sunlf@}tsinghua.edu.cn, zhouchao@kuaishou.com

***Abstract*—Off-the-shelf buffer-based approaches leverage a simple yet effective buffer-bound to control the adaptive bitrate (ABR) streaming system. Nevertheless, such approaches in standard parameters fail to always provide high quality of experience (QoE) video streaming services under all considered network conditions. Meanwhile, state-of-the-art learning-based ABR approach Pensieve outperforms existing schemes but is impractical to deploy. Therefore, how to harmoniously fuse the buffer-based and learning-based approach has become a key challenge for further enhancing ABR methods. In this paper, we propose *Stick*, an ABR algorithm that fuses the deep learning method and traditional buffer-based method. Stick utilizes the deep reinforcement learning (DRL) method to train the neural network, which outputs the *buffer-bound* to control the buffer-based approach for maximizing the QoE metric with different parameters. Trace-driven emulation illustrates that Stick betters Pensieve by 3.5% - 9.41% with an overhead reduction of 88%. Moreover, aiming to further reduce the computational costs while preserving the performances, we propose Trigger, a light-weighted neural network that *determines* whether the buffer-bound should be adjusted. Experimental results show that Stick+Trigger rivals or outperforms existing schemes in average QoE by 1.7%-28%, and significantly reduces the Stick's computational overhead by 24%-61%. Meanwhile, we show that Trigger also helps other ABR schemes mitigate the overhead. Extensive results on real-world evaluation demonstrate the superiority of Stick over existing state-of-the-art approaches.**

I. Introduction

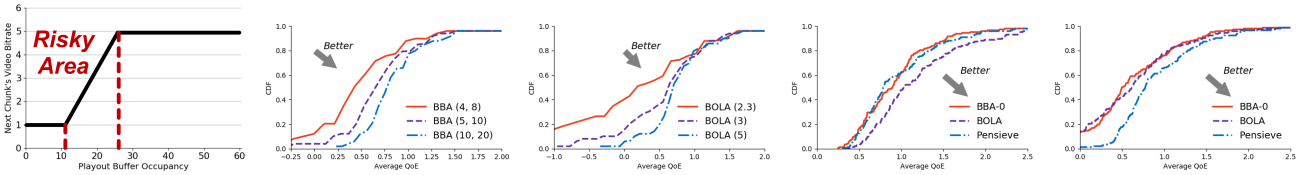
Internet video streaming and downloads are taking a large share of network bandwidth and will grow to more than 82% of all the traffic on consumer Internet by 2022 [1]. Due to the fluctuation of network conditions and diversity of video contents, many adaptive bitrate (ABR) algorithms ([2], [3], [4], [5]) have been proposed to provide video streaming services with high quality of experiences (QoE). Traditional buffer-based ABR algorithms (e.g., BBA [3] and BOLA [6]) select the bitrate of the future video chunk with the fixed buffer-bound or parameters, which not only fail to guarantee the performance under various network conditions but also struggle to meet the different QoE demands (§II). On the contrary, learning-based ABR scheme Pensieve [5] receives state-of-the-art performances via deep reinforcement learning (DRL) method [7], [8]. Specifically, Pensieve is deployed on the server to avoid high computational costs on the client-side. However, in practice, most ABR algorithms are executed in the front-end to avert the extra latency connecting to the back-end [9], [10]. Thus, such ABR policy frameworks ([5], [11]) are theoretically effective but impractical [12].

In light of these concerns, we observe that these two methods are complementary to each other: we can adopt deep learning to enhance the buffer-based approaches, and in turn, leverage buffer-bound to decrease the computational overhead of learning-based approaches, as the buffer-bound carries more redundant information than a single bitrate action. As a result, such methods can make *good but impractical* schemes more *practical*. Hence, this paper is motivated by a simple yet impossible quest: *can we use deep learning method to dynamically adjust the buffer-based approach, aiming to make the proposed scheme perform well under all the considered network conditions, various QoE objectives, and especially, with lower extra costs?* (§II-A)

Based on this question, we list several challenges that attempt to solve them without using deep learning. Unfortunately, despite the abundance of recently proposed approaches [13], [14], recent ABR methods can hardly tackle all of the above challenges due to the respective reasons. To this end, we ask if the *deep learning* could provide the answer that differs from previous methods (§II-B).

In this paper, we propose *Stick*, a novel ABR scheme that fuses the traditional buffer-based and deep learning methods. Unlike traditional buffer-based approaches, Stick passes several underlying metrics, including network status, video features, and QoE parameters into the input(§III-A1). Meanwhile, we take a continuous scalar, represented as a buffer-bound, as the Stick's output. Considering the continuous action spaces, we adopt deep deterministic policy gradient (DDPG) [15], a continuous DRL method, to *train* the model from scratch. In detail, first, Stick outputs the following buffer-bound w.r.t the current states, which includes network status and QoE parameters on demand. Next, the buffer-bound is converted to a chunk map via a linear function. Finally, the client selects the next chunk's bitrate according to the following chunk map and current buffer occupancy. Stick learns the policy by interacting with the ABR environment without any presumptions. Thus, it generalizes a neural network (NN) model, which can achieve high QoE performances under different QoE metrics.

Besides, aiming to reduce the Stick's computational costs



(a) The principle of BBA (b) BBA in different param. (c) BOLA in different param. (d) QoE: $\mu = 0.5$, $\tau = 0.5$ (e) QoE: $\mu = 4.3$, $\tau = 1$

Fig. 1. Comparing the performance of BBA, BOLA, and Pensieve with different parameter settings and QoE metrics (§IV-A4).

effectively, we further propose *Trigger*, a lighted-weighted NN model that activates Stick to update the buffer-bound only if necessary (§III-B). Specifically, Trigger takes past throughput observed, current buffer size and previous buffer-bound as the input. We employ *imitation learning* [16], a high-efficiency learning-based method, to train the Trigger based on the expert actions. To overcome the unbiased distribution of data samples, we utilize *prioritized experience replay* to store expert policies with the following distributions. In addition, we also propose a method that enables other ABR schemes able to use Trigger to reduce the overhead as well.

In the rest of this paper, we set up a trace-driven evaluation and a real-world analysis to validate Stick's performance (§IV). First, we describe the correlations between the number of Stick's outputs and their overall performance. Next, the comparison of Stick and existing buffer-based schemes shows that Stick surpasses recent work by 39.68% to 44.26% on average QoE. We then compare the performance of Stick to existing buffer-based and learning-based ABR schemes. Moreover, the obtained results indicate that Stick with a tiny model size outperforms Pensieve, with the improvements on average QoE of 3.5% to 9.41% under all considered network conditions. Meanwhile, it saves 88% of the cost compared to that of Pensieve. Besides, results also indicate that Stick outperforms existing model-based schemes, with the improvements on average QoE of 10.87% to 30.77%. The additional experiment illustrates that Trigger reduces the Stick's computational costs by 61% and, in the meanwhile, slightly rivals or improves average QoE by 1.7%-4.17% compared with Pensieve. Moreover, we also prove that Trigger can also improve other traditional ABR algorithms by reducing the computational overhead by 24% to 40%. Finally, we implement Stick and evaluate its performances under real-world environments, as expected, yielding solid and reliable results. To sum up, we summarize the contributions as follows:

1) To the best of our knowledge, we are the first to ponder the correlation between the buffer-based and learning-based approaches. Further, we fuse them to enhance each other.
2) We propose Stick, an ABR scheme which is the fusion of deep learning and heuristic methods. Results indicate that Stick can achieve higher performances with smaller model sizes compared with existing ABR schemes.
3) We present Trigger to answer how to use a light-weighted model to accomplish the *cost-reducing task*. Results also illustrate that Trigger can significantly reduce the Stick's computational costs by 61%.

II. Background and Motivation

A. Buffer-based and Learning-based ABR Approaches

ABR algorithm aims to keep video streaming services with high QoE by dynamically picking the next chunk with different bitrates [17] . Buffer-based approach (BBA) [3] is one of ABR algorithms that picks the next chunks' bitrates w.r.t only the current buffer occupancy. In particular, BBA uses throughput estimation in the startup-state to overcome the highly variable throughput. While in the steady-state, BBA uses the fixed buffer-bound $\{B_{min}, B_{max}\}$ to construct *chunk map*, and the *chunk map* is the function for mapping the *buffer occupancy* to the *bitrate*. In this work, as suggested by the original paper [3], we consider a piece-wise linear function as the chunk map to increase the bitrate between lowest bitrate R_{min} and highest bitrate R_{max} of the chunk (see in Figure 1(a)). Notice that the function can be defined as any methods [6]. BBA then selects the future chunks' bitrate based on the client's buffer occupancy and chunk map. In general, BBA is simple yet effective, but fails to perform well in different QoE metrics (Figure 1(d) and 1(e)) and various network conditions (§IV-D). The reason is that the parameters of BBA should be carefully tuned to fit the environment (Figure 1(b) and 1(c)). In contrast, learning-based ABR scheme Pensieve [5] obtains outstanding results via learning from scratch, as it is deployed on the server-side to eliminate the heavy computational cost on the client-side. However, the framework is impractical in practice since most ABRs are light-weighted and often performed on the front-end rather than back-end [3]. Besides, experiments (See Figure 1(d) and 1(e)) show that Pensieve still fails to fit any QoE metrics by using default NN model.

B. Key Ideas and Challenges

Having considered these two methods above, we realize that the learning-based method can assist the buffer-based method to improve the overall performance effectively. Meanwhile, the buffer-based method can also reduce the NN model size and the computational cost because the buffer-bound is flexible and contains redundant information, and it is more tolerant than outputting as a bitrate probability vector. Thus, we address several key challenges.

- ***How to tune the buffer-bound to fit all considered network conditions?*** Ideally, several attempts have been made to tackle the problem from different perspectives. E.g., BOLA [6] theoretically tunes the buffer-bound to keep the system stable, Auto-tuning methods [11], [18] tune parameters w.r.t *a big lookup table* or *a trained NN*. However,

recent approaches either suffer from careful tuning [5] or neglect the storage and computational overhead (§V).

▷ **Our solution.** Motivated by the recent success of deep-learning method [19], [5], we adopt deep reinforcement learning (DRL) method to generalize strategies *without any presumptions*, aiming to adjust the buffer-bound to achieve high QoE performances dynamically. On the other hand, the buffer-bound method has better generalization, which helps NN effectively reduce the model size.

- ***How to determine the state-space and action space for the NN model?*** Recent buffer-based [13] methods consider only the buffer occupancy rather than throughput measurements. What's more, recent auto-tuning method [11] donate throughput features as the input. Ideally, a good ABR method relies on not only the accurate throughput prediction but also other network and video features[20], [4], such as download time and next chunks' sizes. Such metrics are also critical features for ABR algorithms [5].

  ▷ **Our solution.** In our work, we consider the state space into three parts, i.e., network metrics and video content features as well as playback features (§III-A1). Besides, experimental results illustrate that Stick (§IV-B) can control the buffer-bound with one value B rather two values [3].
- ***How to propose an ABR method which can perform well under the QoE metrics with different sets of parameters?*** As demonstrated in Figure 1(d) and Figure 1(e), we find that neither model-based nor learning-based approach is able to handle different types of QoE metrics since these approaches are often optimized towards one QoE function [5], [6].

  ▷ **Our solution.** We donate the QoE parameters as the *goal* and append it into the NN's input (§III-A1). During the training, we *randomly* initialize the parameters from the range of 0.0 to 10.0 so as to let the NN *realize* the correlation between the given QoE parameter and the feedback reward.
- ***How to reduce the computational cost while guaranteeing the overall performance?*** Increased overhead will neglect the increased overall performance if it is significant. E.g., Extensive experiments show that Stick can improve the QoE up to 15.74% compared with the state-of-art ABR approach Pensieve while increasing 45x in terms of the Pensieve's computational cost (§IV-E).

  ▷ **Our solution.** We implement a light-weighted NN model, placed in front of the DRL-based NN, aiming to determine whether it's possible to provide the BBA with a new set of buffer-bound (§III-B). Specifically, we train the NN via imitating the expert value for each step.

In summary, exploring the aforementioned challenges, however, requires not only achieving outperformed performances under different QoE metrics but also reducing the computational cost as much as possible.

III. System Overview

In this section, motivated by the key challenges above, we propose Stick, a ABR approach which is the fusion of buffer-based and learning-based scheme. The Stick's system workflow is illustrated in Figure 2. As shown, the system is mainly composed of i) basic Stick, which leverages DRL to output the buffer-bound for controlling the traditional buffer-based approach (§III-A), and ii) Trigger, which adopts imitation learning to train a light-weighted NN-based change detector for reducing the computational cost (§III-B).

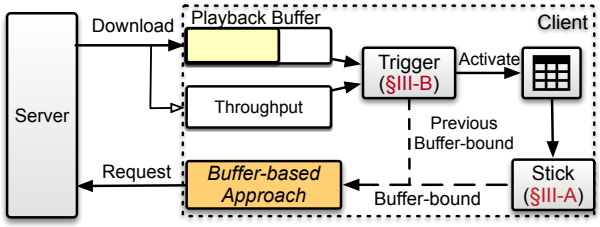


Fig. 2. An overview of Stick.

A. Stick Mechanism

This subsection describes Stick's design (Figure 3), structured according to how it address the three aforementioned challenges: How to design Stick and how to model the *state*, *action*, and *reward*? (§III-A1); How to train Stick? (§III-A2); How to implement Stick? (§III-A3).

1) Design: We consider the lifetime of the ABR process as a Markov Decision Process (MDP), and we formulate Stick via state, action, and reward.

- ***State.*** Stick's learning agent takes the input state of time-slot t $s_t = \{T, d, q, r, b, S, g\}$ into NN, where T means the past throughput measured by client for past k sequence, d represents the download time for past k sequence, q is the previous video bitrate selected; r is the video playback time remaining; b is client's current buffer occupancy; S is a vector that represents the video sizes of the next video chunk; g is the goal represented by two QoE parameters μ, τ. Details are elaborate in §IV-A4.
- ***Action.*** Recall that we aim to control the continuous buffer-bound B rather than directly select discrete bitrates. Thus, our action space is a single scalar which represents the suitable buffer-bound $\mathbf{B} \in [0, \text{Buffer}_{max}]$ for the next chunk.
- ***Reward.*** We use typical linear-based QoE metric (§IV-A4) with different parameters g to implement the reward function.

2) Training Methodology: We leverage DDPG, an algorithm [15] that uses an actor-critic approach via deterministic policy gradient (DPG) [21] to train our model. DPG is composed of an actor network and a critic network. The actor network is updated using Eq.1.

$$\Delta_{\theta_\mu} J \approx E_{st\ p^\beta}[\Delta Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \Delta_{\theta_\mu(s|\theta^\mu)|_{s=s_t}}] \tag{1}$$

Here actor network $\mu(s|\theta^\mu)$ specifies the deterministic policy via a given state, and the critic network $Q(s|a)$ is then updated via Bellman equation (Eq.2).

$$Q^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) Q^\pi(s'). \tag{2}$$

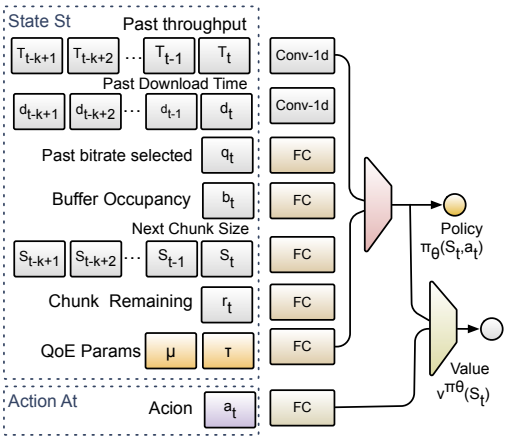


Fig. 3. Stick's NN Architecture Overview. We leverage DDPG to train Stick.

Unlike directly updating the weights of networks, DDPG algorithm *clones* a copy of the actor network $Q'(s,a|\theta^{Q'})$ and critic network $\mu'(s|\theta^{(\theta')})$ to perform target updates. The weights of the networks are updated via slowly matching the learned network via $\theta' = \tau\theta + (1-\tau)\theta'$, where $\tau = 0.001$. The target network is trained slowly for making the network more stable. Besides that, we also leverage the exploration policy to sample from a noise process (Eq.3), where the noise process is chosen to suit the environment. In this paper, we set exploration rate $\sigma = 10$ and exploration decay $\alpha = 10^{-5}$.

$$\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + N. \tag{3}$$

3) Implementation: We use TFlearn [22] to implement the NN, and leverage TensorFlow [23] to construct the training process. Specifically, we set past sequence length $k = 8$. For each feature in the input state, it passes through a conv-1d layer with 32 filters and the kernel size of 4. Finally, the feature maps are concatenated as a tensor. The output of each network is a value in $(0, B_{max})$ and single scalar respectively. We set $B_{max} = 60$ as suggested by [5], [24], $\gamma = 0.99$, actor network's learning rate $\alpha_0 = 10^{-6}$, and the critic network's learning rate $\alpha_v = 10^{-4}$. In addition, we use Adam optimizer [25] to optimize these NNs.

B. Trigger Design

Considering reducing the computational cost while preserving the overall performances, we propose Trigger, which adopts a light-weighted NN model to determine whether it's necessary to activate Stick to update the new set of buffer-bound or not. This section details the design process of Trigger, including its inputs and outputs, training methodology, sample pool module, and implementations.

1) Useless Operation: It's quite apparent that the client will obtain highest QoE if it continuously changes the newer buffer-bound from the Stick. However, we find that there exists a specific computation-wasted situation, that is, when the bitrate selected from the newer buffer-bound is equivalent to the one selected from the past buffer-bound. We treat this form of action as *useless operation*. As shown in Figure 4, we evaluate the distribution of *useless operation* over different network traces. Experimental results illustrate that most of the network traces require only 30% to 40% of average buffer-bound change times. To this end, our key idea for Trigger is to leverage NN model to *learn* whether the states require the useless operation or not.

2) Inputs and Outputs: Recent work [11] has claimed that the network status could be represented by measuring *the mean and variance of throughput*. However, we find that past buffer occupancy is also a critical feature which can demystify the underlying correlations between network status and video chunk sizes. Thus, we use past network throughput observed, the buffer size and current buffer-bound to represent Trigger's inputs. Moreover, we take a two-dims vector into Trigger's output, representing the probabilities of Stick keeping or changing the buffer-bound.

3) Training Methodology: We use imitation learning method to train the Trigger's NN. Imitation learning method reproduces desired behavior according to expert demonstrations [16]. The overall training procedure is described in Alg. 1. Trigger's workflow consists of three parts:

- *Rolling-out the trajectory $A = \{a_0, a_1, \ldots, a_t\}$ with the given sequence $S = \{s_0, s_1, \ldots, s_t\}$;*
- *Offline validating $a_i \in A$ is useless operation and generating a sequence represented as ground truth $Y = \{y_0, y_1, \ldots, y_t\}$, where $y_i = 1$ if it belongs to the useful operation, vice versa;*
- *Updating the Trigger's gradient via behaviour cloning [26] for each step t. In this paper, we use cross entropy error to estimate the loss between expert values and Trigger's outputs.*

Algorithm 1 Trigger Overall Training Procedure

Require: Trained Stick model: Stick; Trigger model π.
Sample Training Batch $B = \{\}$
procedure TRAINING
  Initialize π.
  Get State ABR stateS_t, Trigger state s_t.
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta)$
    if a_t is **not** *useless operation* **then**
      Buffer-bound $A_t = Stick(S_t)$.
    else
      $A_t = A_{t-1}$
    Estimate next chunk's bitrate $\{Q_t, \hat{Q}_t\}$ w.r.t buffer occupancy b_t and two actions $\{A_t, \hat{A}_t\}$;
    Compute expert answer $y_t = \mathbb{1}_{Q_t = \hat{Q}_t}$;
    $B \leftarrow B \bigcup \{s, y\}$ in *Prioritized Experience Replay*.
    Update network via cross entropy loss;
    Get next ABR state S_{t+1} and Trigger State s_{t+1};
    $t \leftarrow t + 1$;
  until Converged

4) Prioritized Experience Replay: Since then, we still have not discussed the distribution of samples. Recent work assumes that data distribution is balanced, which neglects underlying prediction bias. Unfortunately, we observe that there exists a serious imbalance distribution on our Trigger's samples. As illustrated in Figure 4, the distribution of each action is unbalanced, where most network conditions achieve

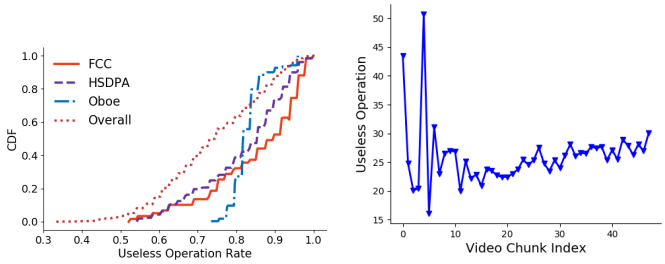


Fig. 4. We test the *useless operation* rate over different network traces including FCC, HSDPA, and Oboe. Results are shown with CDF distributions. Besides, we also evaluate the *useless operation* rate on the entire video *Envivodash3* [27]. Results are collected under various network conditions.

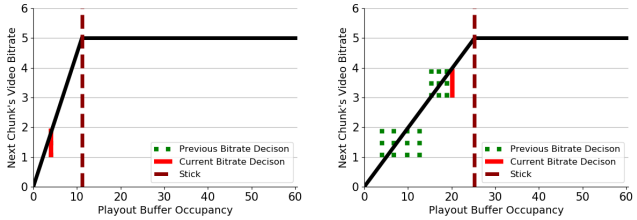


Fig. 5. The principle of how Trigger helps other ABRs. The black slash created by the intersection of the magenta line (buffer-bound) and R_{max} and the origin point is allowed to *pass* as many gray lines (past decisions) as possible. Note that the black slash *must* pass the red line (current decision) at the same time.

the useless operation rate about 83% rather than about 50%. What's more, we find that the probability of useless operation on each chunk is almost equal, which means, the client should use Trigger during the entire session. Hence, we leverage prioritized experience replay[28], [29] to enhance the Trigger's performance.

5) Implementation: Trigger takes the past sequence length $k = 10$ into the NN. For each feature in the input state, it passes through a conv-1d layer with 32 filters and the kernel size of 3. Next, we use a fully connected layer with 32 neurons. The Trigger's output is a 2-dims vector with *softmax* function. We set learning rate $\alpha = 10^{-4}$.

6) Trigger with other ABR schemes: Besides, Trigger can also help other ABR schemes, i.e., Rate-based [2], MPC [4] and Pensieve [5], reduce the computational cost. The key idea is the buffer-bound B only needs to override the bitrate selection R_t for chunk t. The function of mapping R_t to B is allowed to design in any methods, e.g., a simple linear method (Eq.4). We, therefore, consider a fundamental strategy: keep the past chunks' bitrate selected $\{r_0, \ldots, r_{t-1}\}$ as much as possible. To encourage the bound to preserve newer actions, we leverage the discounted factor γ to degrade the value of past sequences. Details are listed in Eq. 4,

$$\max_{B} \mathbf{V} = \sum_{i=0}^{t} \gamma^{t-i} \mathbb{1}_{\{B\}} \tag{4}$$

$$s.t. \quad B \in [\frac{b_t}{R_t + 1} R_{max}, \frac{b_t}{R_t} R_{max}] \tag{5}$$

.

Here t is the sequence length, R_{max} is the maximum bitrate of next chunks, b_t represents current buffer size on chunk t, R_t reflects the bitrate selected by model-based approaches at chunk t. In this paper, we set the hyper-parameter $\gamma = 0.99$. Figure 5 illustrate the fundamental principle: the method aims to generate a line which can *across* previous decisions as much as possible.

IV. Evaluation

In this section, we evaluate Stick and Trigger under various network conditions and compare them with previously proposed ABR approaches.

A. Implementation

1) Experimental Testbed Setup.: To better validate ABRs, out work is composed of two experiments: trace-driven offline emulation and real-world experiment.

▷ **Trace-driven offline emulation.** In the past few years, there are many issues for the researchers to implement a faithful simulator to emulate the ABR process in the offline environment, e.g., TCP-slow start [3], which causes the failure of bandwidth estimation; lack of various real-world network traces, which limits the effect of the offline simulator. Fortunately, several faithful attempts [5], [14], [11] have been proposed make validation process easier. Thus we utilize Pensieve's trace-driven emulation environment, including Mahimahi [30] and Park [31], to evaluate Stick.

▷ **Real-world Deployment.** We also establish a full-system implementation, which consists of a video player, an ABR server and an HTTP content server. On the server-side, we deploy an HTTP Server with TCP-slow-start restart disabled as suggested by Pensieve [5]. On the client-side, we modify `Dash.js` [27] to implement our video player client and the Trigger module. Further, we place Stick as a service on the client locally.

2) Video Datasets.: To better enhance the Stick's generalization performance, we train Stick via a video dataset provided by Huang et al. [32]. The dataset 86 complete videos, with 394,551 video chunks. In details, the video is encoded by the H.264 codec at video bitrates in the range of $\{0.3, 0.75, 1.2, 1.85, 2.85, 4.3\}$ Mbps, where each chunk is 4 seconds. Meanwhile, for validating Stick, we use *Envivio-Dash3* [33], which is commonly used in recent work [5], [24], [11]. Furthermore, we also pick an additional video set from video dataset [32] for testing. The total length of the video is 212 seconds, and the video is divided into 53 chunks.

3) Network trace datasets.: We use a large corpus of network trace dataset (i.e., Kwai dataset) provided by Kuaishou [34]. The dataset is collected in the number of 865,507 traces from 9,941 users, totally 46 days from various network conditions, including wired, WiFi and cellular network, and so forth. Specifically, we randomly pick the 10% of the dataset and divide them into two parts, 80% of the database for training and 20% for testing. What's more, we also leverage a corpus of public datasets for validating Stick. Detailing the validation set, that includes: *HSDPA* [35]:

a well-known 3G/HSDPA network trace dataset (228 traces, 1s granularity); *FCC* [36]: a broadband dataset (486 traces, 1s granularity); *Oboe* [11] (428 traces, 1-5s granularity): a trace dataset collected from vaious network conditions.

4) QoE Metrics.: In this paper, followed by recent work [4], [5], [24], [11], we use the general QoE metric QoE_{lin}, the linear mapping formula which was used by MPC [4], to evaluate Stick. The metric is defined as:

$$\text{QoE} = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \tau \sum_{n=1}^{N-1} \left| q(R_{n+1}) - q(R_n) \right|, \tag{6}$$

where N is the total number of chunks during the session, R_n represents the each chunk's video bitrate, T_n reflects the rebuffering time for each chunk n, $q(R_n)$ is a function that maps the bitrate R_n to the video quality perceived by the user. μ and τ is the coefficient which control the priority of the given metric. Followed by recent work [5], [11], [24], we set $q(R_n) = R_n$, $\mu = 4.3$ and $\tau = 1.0$ as the basic QoE baselines. Moreover, we also compare Stick with Different QoE metrics in §IV-F.

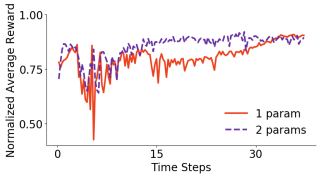
5) Training Time.: We train Stick on a desktop with i7-8700k CPU in 12 cores, 32GB RAM and a GTX1080Ti GPU card. Training process lasted approximate 40000 steps, or almost 3 hours to achieve a stable result. We also train Trigger on the same settings and the training time lasts about 2 hours.

6) ABR Baselines.: In this paper, we select several representational ABR algorithms from various type of fundamental principles:

1) ***Rate-based* [2]:** uses harmonic mean of past five throughput measured as future bandwidth. It then selects the highest bitrate but lower than the future bandwidth estimated.
2) ***Buffer-based* [3]:** dynamically chooses next chunk bitrate according to the buffer occupancy. In this paper, we set the buffer bound $B = 5, 10$ as suggested by the authors.
3) ***BOLA* [6]:** turns the ABR problem into a utility maximization problem and solve it by using the Lyapunov function. It's a buffer-based approach. We use BOLA provided by the authors [14].
4) ***HYB* [11]:** predicts the throughput only over a horizon of 1 future chunk and selects the next bitrate using the heuristic method. We set discounted buffer factor $\beta = 0.25$ as mentioned by the authors.
5) ***MPC* [4]:** maximizes the QoE objectives by jointly considered the buffer occupancy and throughput predictions. We implement *RobustMPC* by ourselves.
6) ***Pensieve* [5]:** utilize DRL to select bitrate for next video chunks. Pensieve takes the former network status as states and reinforces itself through the interaction with the faithful offline simulator. We use the pre-trained Pensieve model provided by the authors.

B. Stick with Different Outputs

In this part, for proving the effectiveness of Stick, we design an experiment to figure out how necessary the Stick with two sticks is. During the training process, we compare the average reward of Stick using two parameters. Results are evaluated under the Oboe trace dataset every 100 training steps. Experimental results are summarized as the normalized average reward in Figure 6(a). As shown, although Stick with two parameters converges faster than the one with a single parameter, the performance of these two schemes has no obvious difference. Moreover, details are listed in Figure 6(b), and we find that the scheme using one parameter also betters the other one with the improvements of average QoE on 2.8%. As a result, we represent Stick with a single output as the fundamental structure of the system.

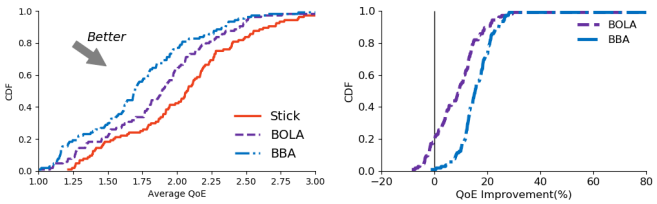


(a) Learning curves

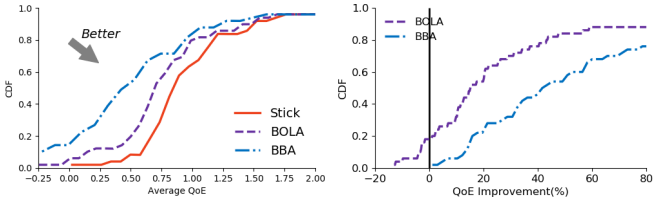
ABRs	Size	Avg. QoE
BBA	-	1.90
Pensieve	128	2.12
Stick@2	64	2.14
Stick@1	**32**	**2.20**

(b) Statistics Details

Fig. 6. Comparing the performance of Stick using 1 parameters with Stick using 2 parameters. Results are evaluated on the oboe network traces.



(a) Kwai Network Dataset

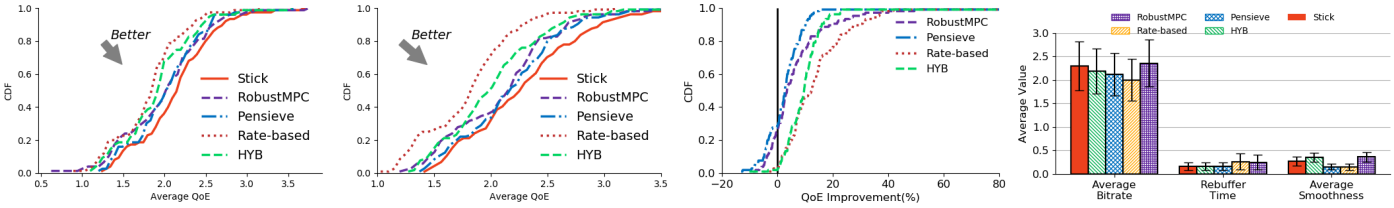


(b) HSDPA Network Dataset

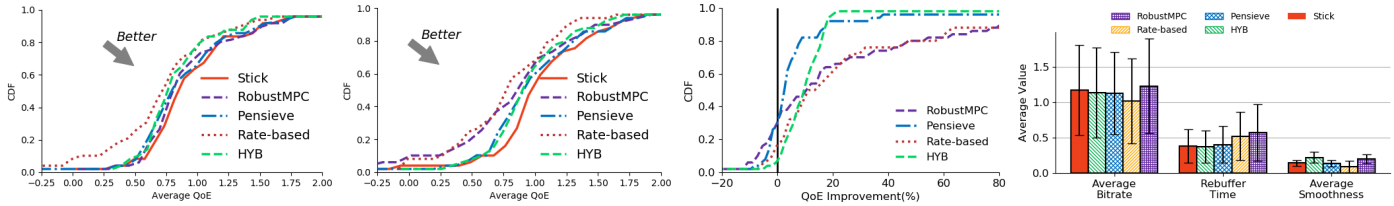
Fig. 7. Comparing average QoE of Stick with BBA on the QoE metrics on the HSDPA and Kwai network dataset.

C. Stick vs. Buffer-based Approach

In this experiment, we demonstrate that Stick, leveraging deep learning method, performs better than the buffer-based approaches. We use offline trace-driven emulation to evaluate Stick, BOLA, and BBA under HSDPA and Kwai network dataset. CDF distributions of proposed methods are shown in Figure 7(a). The result indicates that Stick outperforms existing buffer-based approach on the Kwai dataset, with the improvements on average QoE of 9.63% in terms of BBA and 18.5% in terms of BOLA respectively. Besides, we find that Stick improves the 95th percentile average QoE by 19.83% compared with BBA. Further, Figure 7(a) also illustrates the CDF of the improvement on QoE of Stick over buffer-based approaches. As expected, we observe that Stick improves the performance for 100% sessions on BBA, and almost 80% of



(a) Kwai network dataset.



(b) HSDPA network dataset.

Fig. 8. Comparing Stick with existing ABR approaches under **different videos** (§III-A3) and network traces (i.e., Kwai and HSDPA network datasets). Results are illustrated with CDF distributions, QoE improvement curves and the comparison of several underlying metrics (§IV-A4).

sessions on BOLA. What's more, Figure 7(b) compares Stick with buffer-based approaches on the HSDPA dataset, and we also illustrate that Stick improves 44.26% compared with BBA and increases 25.93% in terms of BOLA.

D. Stick vs. Existing ABR schemes

In this experiment, we use trace-driven offline emulation to compare the performance of Stick against several existing ABR algorithms, including model-based ABR schemes (i.e., Rate-based, HYB, and MPC) and learning-based ABR scheme (i.e., Pensieve), over Kwai and HSDPA network dataset. Specfically, we evalaute the proposed schemes with different video traces as described in §IV-A2.

▷ **Stick vs. Pensieve.** Figure 8 illustrates the CDF of QoE for Stick and Pensieve, and we find that Stick improves the average QoE of 9.41% over Kwai dataset and 3.5% over HSDPA dataset respectively. Note that Stick only utilize *one-quarter number of neurons* that of Pensieve. Meanwhile, we also analyze the CDF of the QoE percentage improvement for Stick over Pensieve. Exterimental results indicates that Stick performs better than Pensieve on 77% of the sessions. Furthermore, we report the average performance of each scheme with three underlying metrics. We can see that comparing the metrics with Pensieve, Stick improves the average bitrate by 8.49%, decreases the average rebuffering time by 4.92%, and slightly increases average bitrate change by 3.47%.

▷ **Stick vs. Other Schemes.** Figure 8 shows the CDF distribution of average QoE performances for each ABR algorithms over different network traces. As expected, Stick outperforms existing schemes, with the improvements on average QoE for the entire session of 9.5% to 25.86%. Moreover, we also report the utility from the average bitrate, rebuffering time as well as switching bitrates. The obtained results indicate that compared Stick to other schemes, Stick always achieve higher average bitrate with lowest rebuffering and bitrate changes. Note that Stick fails to obtain the highest bitrate among the candidates, but results the highest average QoE. That means, Stick has generalized a strategies to fit the network environment rather than blindly increase one of the underlying metrics. Same conclusion are also demonstrated in Pensieve [5].

TABLE I
STICK DEEP DIVE: SWEEPING THE NUMBER OF CNN FILTERS AND HIDDEN NEURONS FOR PENSIEVE'S NN MODEL.

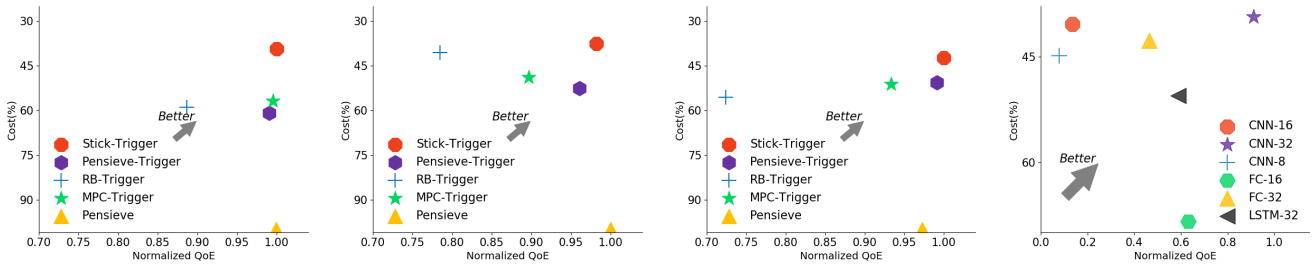
N	Pens. [5]	16	32	64	128	1024
Avg.QoE	2.12	1.97	2.20	2.18	2.22	2.24
Size(MB)	1.0	0.07	0.23	0.47	1.0	80.67
KFLOPs	298	16	37	99	296	13381
KFLOPs Per.(%)	-	5	12	33	99	4494

E. Stick Ablation Study

In this experiment, we investigate how Stick performs under different hyper-parameters. We start by sweeping a range of NN parameters to evaluate each performances, and we set N in range $\{16, 32, 64, 128, 1024\}$, where N means the number of filters in CNN layers and the number of neurons in fully connected layers. Results are evaluated under the Oboe dataset.

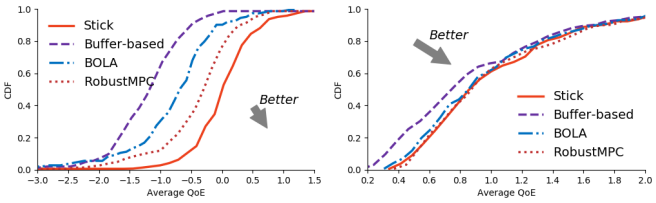
▷ **Stick Deep Dive.** As demonstrated in Table I, we observe that with the same NN architecture, Stick exceeds Pensieve by using only 25% of Pensieve's parameters, with the average QoE improvements of 1.4%. Further, results also illustrated that with the increase of model size, the performance of Stick will consistently improve up to 5.66% compared to Pensieve. Note that the Stick which is sized more than 1M will no longer support to deploy on the client-side because it's somewhat huge for the cost on both computing and downloading.

▷ **Stick Overhead.** Meanwhile, we also compute the number of floating-point operations (FLOPs) for each model [37], as illustrated in Table I. We find that Stick saves 88% of the computational cost compared to Pensieve, and such advantages lasts up to Stick-128. In general, comparing the model size of Stick with Pensieve, we believe that using the fusion of



(a) Oboe (b) FCC (c) HSDPA (d) Best NN Selection

Fig. 9. We evaluate Trigger with Stick and other ABR schemes over different network traces including FCC, HSDPA and Oboe. Results are shown as performance-cost plot. Besides, we also discuss the best architecture for Trigger.



(a) QoE: $\mu = 6, \tau = 6$ (b) QoE: $\mu = 0.5, \tau = 0.5$

Fig. 10. Comparing Stick with several ABR schemes, including BBA, BOLA, RobustMPC and state-of-the-art learning-based approach Pensieve over different QoE metrics. Results were collected on the HSDPA dataset.

domain knowledge and deep learning will significantly reduce both computational and storage overhead.

F. Stick with Different QoE metrics

In this experiment, we only train Stick *once* and evaluate it with different QoE parameters. It's quite challenging since both buffer-based [3], [6] and learning-based ABRs [38], [5] seldom achieve the goal under various QoE. Results are shown in Figure 10, and we observe that the performance of Stick also surpasses that of recent ABR schemes under different QoE parameters. In particular, Stick betters traditional buffer-based scheme on average QoE of 45.8% in which QoE: $\mu = 6, \tau = 6$. Such positive results prove the effectiveness of taking QoE parameters as the input (§III-A1).

G. Trigger Evaluation

In this part, we tried to demonstrate the importance of Trigger for the Stick system. The result indicates that Trigger works well not only on Stick but also on other ABR schemes. We then figured out the best NN architecture for Trigger to the best of our knowledge.

▷ **Stick+Trigger Evaluation.** We setup an experiment to better understand how Trigger performs. In details, we evaluate average QoE metrics and useless operations on Trigger+Stick in the same model size. To better understand how does Trigger help traditional ABR schemes reduce the computational cost, we validate three representative algorithms, i.e., rate-based approach, MPC, and Pensieve. Results are collected under Oboe, FCC, and HSDPA network conditions. Figure 9(a) illustrates, using Trigger will significantly reduce the overhead of Stick, with the average cost decreases of 39.3% to 61.0%. At the same time, it also improves the average QoE of 1.70% to 4.17% compared with Pensieve. In particular, we observe that Trigger even improves the overall QoE of traditional heuristic approaches, i.e., rate-based approach and MPC, with the improvements on average QoE of 9.3% to 12.3%. The reason for this is that Trigger uses past buffer occupancy that conventional rate-based approach neglects. Besides, Trigger also helps MPC avoid switching bitrates frequently so as to achieve higher QoE performances. As expected, the results on Figure 9(b) and Figure 9(c) also illustrate the same point. Especially, Trigger+Pensieve even works better than the original Pensieve, by 2.6% to 3.4% on average QoE metric. The reason is that Trigger further considers the correlation between throughput and buffer occupancy from the global perspective so that Pensieve can reduce the bitrate change by further utilizing the buffer.

▷ **Best NN architecture selection.** To better understand why Trigger performs well, we compare the NN architecture from Trigger to the following network architectures which collectively represent the architecture candidates. The network architecture candidates are simply listed as follows:

1) **Fully Connected**: $\texttt{FC}^1_N \rightarrow \texttt{FC}^1_{N/2} \rightarrow \texttt{FC}^2_N$
2) **LSTM**: $\texttt{LSTM}^1_N \rightarrow \texttt{SELF-ATTENTION}^1_N$ [39]
3) **1D-CNN***: $\texttt{CONV1D}^{0\cdots1}_N, \texttt{FC}^1_N \rightarrow \texttt{MERGE}^1 \rightarrow \texttt{FC}^2_N$.

* *Trigger's NN architecture.*

Here N represents the counts of hidden units in a fully connected layer or the channel number in the conv layer and we set $N = \{8, 16, 32\}$. In this experiment, we train and validate the following under same network traces and videos. Results are demonstrated in Figure 9(d), and indicate that 1D-CNN NN architecture succeeds in improving the capacity, with increases in average QoE of 0.46% to 1.37% and decreases in average costs of 1.0% to 28%. Through the experiment, We also confirm that there is no obvious difference between these architectures in terms of operational efficiency.

H. Stick in the Real-world

The final challenge of Stick is to meet the real world network conditions. In this experiment, we evaluate Stick, Stick+Trigger as well as Pensieve under several types of network connections, and the detail of each link is listed as follows:

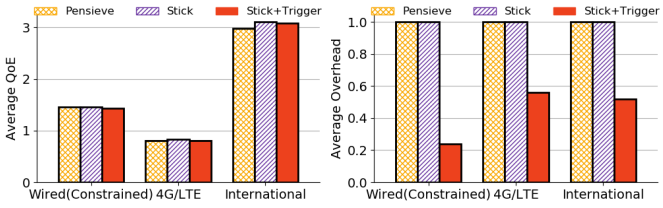


Fig. 11. Comparing the average QoE of Stick with Pensieve under various real-world network conditions.

1) A wired network link from *Shanghai* to *Beijing*, where Round Trip Time (RTT)=36ms, packetloss¡2%. In order to validate how Trigger works, we fixed the the server's bandwidth to 2Mbps.
2) A 4G/LTE network link from *QingDao* to *Beijing*, where RTT=58ms, packet loss=3.2%.
3) An international network connection from *Singapore* to *Beijing*, in which RTT=315ms, Loss≈8%.[1].

Specifically, for each round, we randomly select a scheme from the following candidates and summarize the bitrate selected and rebuffering time for each chunk. Each link takes about 2 hours, and 6 hours in total. Results are reported with the average QoE of each session. As illustrated in Figure 11, as expect, we find that Stick performs comparable or better than Pensieve. Especially, Stick outperforms Pensieve on the international link, with the improvements on average QoE of 4.38%. The reason is that Stick leverage a buffer-bound to keep the ARB system, and the buffer-bound contains more redundant network information as mentioned before (§II). Thus, such scheme has the abilities to handle the *high throughput but high variance network conditions* such as international link. What's more, we also observe that Trigger reduces the Stick's average computational cost by 42% to 76%. In particular, Trigger saves almost 80% overhead on the wired network link. Such positive result indicates that Trigger have learned the strategies to avoid redundant decisions and perform well under the non-stationary network. Note that wired network link with fixed bandwidth are widely used in the real-world.

V. Related Work

Recent ABR algorithms are mainly composed of three types, i.e., model-based, learning-based, and auto tuning-based.

▷ **Model-based Approaches.** FESTIVE [2] estimates future throughput via the harmonic mean of the throughput measured for the past five chunk downloads. However, due to the lack of throughput estimation method currently, these approaches still result in poor ABR performance. Then, many approaches are designed to select the appropriate high bitrate next video chunk and avoid rebuffering events based on playback buffer size observed. BBA [3] proposes a linear criterion threshold to control the available playback buffer size. BOLA [6] turns the ABR problem into a utility maximization problem and solve it via the Lyapunov function. However, typically buffer-based approach fails to tackle the long-term bandwidth fluctuation problem [17]. Hence, mixed approaches [4] select bitrate for next chunk by adjusting its throughput discount factor based on past prediction errors and predicting its playback buffer size. Nevertheless, these approaches require careful tuning because they rely on parameters that are quite sensitive to network conditions, resulting in poor performance in unexpected network environments [5].

▷ **Learning-based ABR Algorithms.** Recently, several attempts have been made to optimize ABR algorithm based on reinforcement learning method due to the difficulty of tuning mixed approaches for handling different network conditions. D-DASH [38] uses Deep-Q-learning method to perform a comprehensive evaluation based on state-of-the-art algorithms. Tiyuntsong [41] optimizes itself towards a rule or a specific reward via the competition with two agents under the same network condition. Pensieve [5] utilizes DRL to *learn* an ABR algorithm without any presumptions. Comyco [32] trains the ABR policy via imitating expert trajectories, which can avoid redundant exploration and make better use of the collected samples. However, these approaches only focus on improving the performance but seldom consider the overhead.

▷ **Auto tuning-based ABR Algorithms.** CS2P [42] assume that throughput factors can be efficiently captured by Hidden-Markov-Model (HMM), then they optimize the model on the cloud with huge amounts of data. Oboe[11] attempts to place a dictionary, which mapping the throughput status $\{\mu, \sigma\}$ to the optimized traditional ABRs' ([4], [6]) parameters, on the cloud for assisting traditional algorithms to achieve higher performances over different network conditions. Erudite [18] adopts NN, trained by the Bayesian Optimization Algorithm, to continuously provide the controller with near-optimal parameters. However, such schemes suffer from either huge computational costs or large waste of storage space. Besides that, deploying a lookup table on the client-side is also impractical [12].

VI. Conclusion

This work starts with an intuitive yet impossible idea: *merge* the traditional buffer-based and learning-based ABR scheme to implement a fusion approach, since we observe that those two methods are complementary to each other. To that end, we propose Stick, a novel ABR scheme which jointly improves the performance and reduces the cost. Unlike previous approaches, Stick utilizes DRL to dynamically adjust the buffer-bound, and controls the ABR algorithm w.r.t the current buffer-bound. Results show that Stick outperforms existing schemes, with the improvements of 9.41%-44.26%. Note that Stick only uses 12% computational costs that of state-of-the-art ABR scheme Pensieve. Meanwhile, we also propose a tiny model named Trigger to further reduce the overhead while guaranteeing the overall performance. Experimental results illustrate that Trigger betters existing schemes and reduces 61% on computational costs. At the same time, Trigger also helps traditional ABR algorithms reduce the overhead up to 40%. In conclusion, Stick is a feasible ABR framework which can harmoniously fuse the buffer-based and learning-based.

[1]In this experiment, we use TCP-BBR [40] as the basic TCP congestion control algorithm. Notice that the packet loss metric is collected by *Ping*.

ACKNOWLEDGEMENT

We thank the anonymous INFOCOM reviewer for the valuable feedback. We also thank Luca De Cicco for fruitful discussions in terms of the Trigger module in $Stick$. This work was supported by the National Key R&D Program of China (No. 2018YFB1003703), NSFC under Grant No. 61936011 and No. 61521002, Beijing Key Lab of Networked Multimedia, and Kuaishou-Tsinghua Joint Project (No. 20192000456).

REFERENCES

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2016-2021," 2017.

[2] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," *IEEE/ACM Transactions on Networking (TON)*, vol. 22, no. 1, 2014.

[3] 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," *ACM SIGCOMM 2014*, vol. 44, no. 4, 2014.

[4] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *ACM SIGCOMM 2015*. ACM, 2015.

[5] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *ACM SIGCOMM 2017*. ACM, 2017.

[6] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *INFOCOM 2016, IEEE*. IEEE, 2016.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[9] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 157–168.

[10] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE multimedia*, vol. 18, no. 4, pp. 62–67, 2011.

[11] Z. Akhtar and et. al., "Oboe: auto-tuning video abr algorithms to network conditions," in *ACM SIGCOMM 2018*. ACM, 2018.

[12] H. Mao, S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh, and E. Bakshy, "Real-world video adaptation with reinforcement learning," in *ICML 2019 Workshop*, 2019.

[13] Y. Zhang and Y. Liu, "Buffer-based reinforcement learning for adaptive streaming," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2569–2570.

[14] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: improving bitrate adaptation in the dash reference player," in *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 2018.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[17] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A survey on bitrate adaptation schemes for streaming media over http," *IEEE Communications Surveys & Tutorials*, 2018.

[18] L. De Cicco, G. Cilli, and S. Mascolo, "Erudite: a deep neural network for optimal tuning of adaptive video streaming controllers," in *MMSys 2019*. ACM, 2019.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, 2016.

[20] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha, "Can accurate predictions improve video streaming in cellular networks?" in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 2015, pp. 57–62.

[21] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[22] Y. Tang, "Tf. learn: Tensorflow's high-level module for distributed machine learning," *arXiv preprint arXiv:1612.04251*, 2016.

[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[24] P. G. Pereira, A. Schmidt, and T. Herfet, "Cross-layer effects on training neural algorithms for video streaming," in *NOSSDAV 2018*. ACM, 2018.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," *arXiv preprint arXiv:1805.01954*, 2018.

[27] "Dash industry forum — catalyzing the adoption of mpeg-dash," 2019. [Online]. Available: https://dashif.org/

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[30] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for {HTTP}," in *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, 2015, pp. 417–429.

[31] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, R. Addanki, M. Khani, S. He *et al.*, "Park: An open platform for learning augmented computer systems," in *ICML 2019 Workshop*, 2019.

[32] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *Proceedings of the 27th ACM International Conference on Multimedia*. ACM, 2019, pp. 429–437.

[33] "reference client 2.4.0-2016," 2016.

[34] "Kuaishou," 2019. [Online]. Available: https://www.kuaishou.com

[35] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *MMSys 2013*. ACM, 2013.

[36] M. F. B. Report, "Raw data measuring broadband america 2016," https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, 2016, [Online; accessed 19-July-2016].

[37] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[38] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-dash: A deep q-learning framework for dash video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703–718, Dec 2017.

[39] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[40] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[41] T. Huang, X. Yao, C. Wu, R.-X. Zhang, and L. Sun, "Tiyuntsong: A self-play reinforcement learning approach for abr video streaming," *arXiv preprint arXiv:1811.06166*, 2018.

[42] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *ACM SIGCOMM 2016*. ACM, 2016.