Tiyuntsong: A Self-Play Reinforcement Learning Approach for ABR Video Streaming

Tianchi Huang, Xin Yao, Chenglei Wu, Rui-Xiao Zhang, Lifeng Sun
Dept. of Computer Science and Technology, Tsinghua University, Beijing, China
{htc17,yaox16,wucl18,zhangrx17}@mails.tsinghua.edu.cn, sunlf@tsinghua.edu.cn

ABSTRACT

Existing reinforcement learning (RL)-based adaptive bitrate (ABR) approaches outperform the previous fixed control rules based methods by improving the Quality of Experience (QoE) score, as the QoE metric can hardly provide clear guidance for optimization, finally resulting in the unexpected strategies. In this paper, we propose *Tiyuntsong*, a self-play reinforcement learning approach with generative adversarial network (GAN)-based method for ABR video streaming. Tiyuntsong learns strategies automatically by training two agents who are competing against each other. Note that the competition results are determined by a set of rules rather than a numerical QoE score that allows clearer optimization objectives. Meanwhile, we propose GAN Enhancement Module to extract hidden features from the past status for preserving the information without the limitations of sequence lengths. Using testbed experiments, we show that the utilization of GAN significantly improves the Tiyuntsong's performance. By comparing the performance of ABRs, we observe that Tiyuntsong also betters existing ABR algorithms in the underlying metrics.

ACM Reference Format:
Tianchi Huang. 2019. Tiyuntsong: A Self-Play Reinforcement Learning Approach for ABR Video Streaming. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recent years have witnessed a rapid growth of online video streaming applications and services [5]. To achieve smooth video playback under various network conditions, modern client-side video player often adopts ABR algorithm to dynamically determine the bitrate of next video chunk to download for achieving high QoE score including high video bitrate, low rebuffering, etc.. Most of the approaches, such as throughput-based[13, 16], buffer-based[12, 25] and mixed schemes[24, 30] often employ fixed control rules which determine future video bitrates via carefully tuned strategies and thresholds. However, these approaches are often designed with strong assumptions of the real-world network conditions and heavily rely on the fine-tuned parameters, which result in sensitivities to network conditions and unexpected performances. To address these problems, researchers [9, 18] have proposed to leverage the reinforcement learning (RL) method to *learn* an algorithm from scratch without any network presumptions, and meanwhile the state-of-the-art deep reinforcement learning (DRL) scheme Pensieve [18] outperforms existing ABRs in some settings. These work tries to optimize a neural network towards a better QoE score, in which the fine-tuned parameters have significant impacts on the performance. However, we observe that RL-based method often obtains high QoE score via some tricks due to the lack of guidance for optimization in QoE. As a result, despite its abilities to obtain higher numerical QoE scores, such training schemes may generate a strategy that doesn't meet the basic rules of the ABR algorithm.

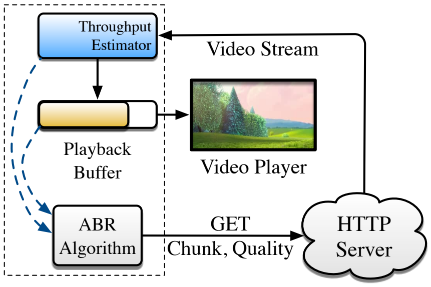

Figure 1: An overview of ABR video streaming.

Our key idea is to regard the reward as a rule instead of QoE metrics. The rule is allowed to use any methods, such as a logistic and AI method, to identify the better one from two candidates. Unlike previous work, the rule will highlights the priority of optimization to avoid occurring unexpected strategies. Based on this idea, we propose *Tiyuntsong*[1], a self-play RL method with GAN for ABR video streaming. Tiyuntsong trains two agents simultaneously for generating a well-performed ABR algorithm under different network conditions. In detail, Tiyuntsong first uses two agents to provide the video streaming service on the same network condition and the video content respectively. Next, it leverages the rule to determine the winner. Finally, it assigns the reward of each agent as *{win:1, lose:0}* and updates the two agents' gradients. In brief, Tiyuntsong approaches a Nash equilibrium via the self-play method, whereas traditional RL methods diverge.

Besides, we further present *GAN Enhancement Module*, a GAN-based method to extract hidden features from past status that facilitate Tiyuntsong store the information without the limitation of

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] Tinyuntsong: Also named as *Cloud Ascending Ladder*, a qinggong skill in the Chinese wuxia novel *The Heaven Sword and Dragon Saber* by Jin Yong. The skill enables the user to travel at high speeds and leap to extreme heights by stepping one foot on the other one.

sequence length. In short, we design two neural networks, one is used for generating future hidden feature based on current state and hidden feature, and the other one is designed for estimating the probability of whether the hidden feature comes from the winning sample or not.

In the rest of our paper, we first discuss Tiyuntsong's neural network architecture. After that, we collect a large corpus of network traces from alternative public datasets for training and validating. Next, we leverage Elo-Rating [8], a classic rating-based system to compute the performance of Tiyuntsong via winning percentage. Using a testbed experiment we prove the importance of GAN Enhancement Module. In all consider scenarios, Tiyuntsong outperforms the existing ABR approaches in the underlying metrics of ABR including bitrate and rebuffering as well as smoothness.

To sum up, our contributions are as follows:

- We figure out the weakness of RL-based ABR algorithms and suggest a novel sight to redefine the reward metric for them: Use logistic rules to describe QoE instead of QoE metrics.
- We are the first to use self-play RL method to tackle the ABR video streaming problem. Results prove that Tiyuntsong not only avoids deviating from the fundamental rule but also betters recent work.
- Despite the abundance of recently proposed methods [7, 15], RL-GAN methods focus on improving the imitation rather than preserving the useful information. Results also indicate that the model significantly improves the RL's performance.

2 BACKGROUND AND MOTIVATION

In this section, we start with introducing the ABR's definition in finite words. We then explain our motivation of this paper, that is, no matter how precisely and carefully the QoE tune, traditional RL-methods cannot exactly provide a result that the users really desire. Finally, we propose a novel sight: Use the self-play method to tackle this dilemma.

2.1 Background on ABR

Due to the rapid development of network services, watching video streaming online has become an upcoming trend. Today, adaptive video streaming, such as HLS (HTTP Live Streaming) and DASH, an algorithm that dynamically selects video bitrates via network conditions and client's buffer occupancy, is the predominant form of video delivery. The traditional video streaming architecture is shown in Figure 1, which consists of a video player client with a constrained buffer length and an HTTP-Server or Content Delivery Network (CDN). The video player client decodes and renders video frames from the playback buffer. Once the streaming service starts, the client fetches the video chunk from the HTTP Server or CDN orderly by an ABR algorithm, and, in the meanwhile, the ABR algorithm, implemented on the client side, determines the next chunk N and next chunk video quality Q_N via throughput estimation and current buffer utilization.After finished to play the video, several metrics, such as total bitrate b, total re-buffering time r and total bitrate change s will be summarized as a QoE metric to evaluate the performance. Thus, achieving a high QoE score for video streaming has become a major challenge for ABR algorithms.

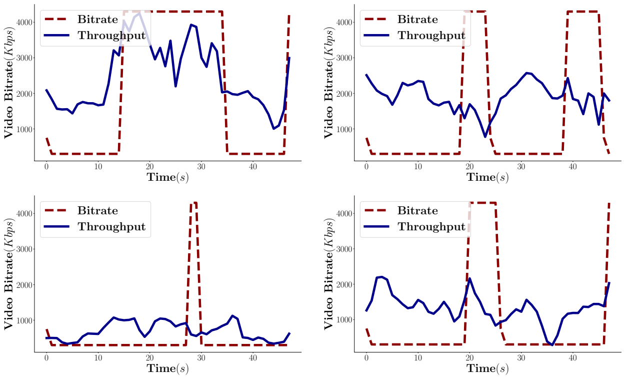

Figure 2: This group of figures shows the drawback of traditional RL-based ABR methods: After trained Pensieve with default parameter settings about 7 days, we observed that: 1) Its policy was simple and effective; 2) It still maintained a high QoE score. However, it's clear that the proposed method deviates from the basic rules of ABR.

To overcome this challenge, most traditional ABR algorithms leverage time-series prediction (throughput-based) or automation control method (buffer-based) to make decisions for the next chunk. Moreover, [18] suggests that traditional fixed control rules methods require careful tuning and will achieve bad performances in the circumstance which is different from the assumption. As a result, traditional ABR algorithms perform well in pre-assumption and specific network conditions but hard to keep its performance in various network environments.

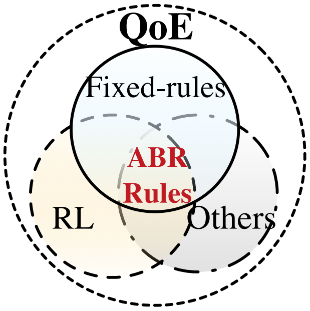

Figure 3: Traditional RL method's Trap: A QoE metric can evaluate several ABR algorithms, but the generated algorithm may deviate from the basic rules of the ABR if it blindly improves QoE score.

2.2 The Trap of Traditional RL-based Method

Considering the ABR process as a Markov Decision Process (MDP), in recent years, many schemes have been proposed to learn ABR algorithms via RL method [4, 9, 18]. Despite the outstanding performances that RL-based ABR algorithms achieve, these schemes suffer from a key limitation: They optimize their neural network via enhancing QoE scores. However, achieving a high QoE score doesn't

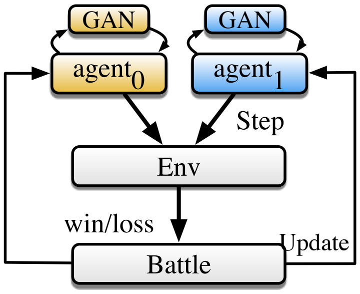

Figure 4: Tiyuntsong overview

necessarily mean as generating an excellent algorithm. For example, the experimental results of state-of-the-art RL-based scheme Pensieve[2] is illustrated in Figure 2. Though Pensieve gains much higher QoE score than any other approaches, it seems like to explore some *tricks* for obtaining higher reward score instead of generating a well-performed algorithm. While Pensieve converges, its policy can be finally concluded as *Fetching lowest bitrates till the buffer has enough space and time to fetch highest bitrates*. That's because, in most cases, which contains only one factor (e.g. playing Atari games), the RL-based model will be trained as expected. When facing the problem infected by multiple factors, recent work [25, 30] defines a formula with a weighted sum of several underlying metrics as the reward (e.g. QoE in ABR problems). However, as demonstrated in Figure 3, we find that more than one rule can be generalized to represent the same QoE formulation, vice versa. Thus, QoE driven RL-based approaches are able to produce a relative good numerical reward, but they may also provide users with unexpected and unstable viewing experience. This problem imposes a critical challenges for RL-based ABR algorithm.

To tackle this problem, we find that AlphaZero [23], a scientific breakthrough in AI, is trained solely based on self-play RL and is periodically matched against several games. We, therefore, consider following AlphaZero to train the algorithm via self-play RL. However, static games with incomplete information (e.g., Training ABR with two agents) is much dissimilar and more complicated than dynamic games with complete information such as Go. Hence, how to design proper model and how to define a suitable reward representation for ABR have become new challenges.

3 TIYUNTSONG'S MECHANISM

In this section, we provide the design steps and the implementation details of Tiyuntsong. As stated before, conventional RL is not a suitable scheme to solve the complex reward function problem where its reward is computed as a linear combination of multiple factors. We, therefore, suggest a novel sight to describe the reward function: only to represent the reward as *win* or *loss* rather than an actual reward score. Following this sight, we propose Tiyuntsong, a self-play with RL method which learns an algorithm automatically based on the only *win* and *loss* signals. As illustrated in Figure 4, two agents competes for each other in the same environment and then update their network based on the competitive result.

3.1 The Design of Agent

We first initialize two agents $\mathbf{A_0}$ and $\mathbf{A_1}$. Note that Tiyuntsong's neural network architecture is quite different from the common RL's representation due to the distinctiveness of the ABR task, and it will be discussed in implementation. The rest of the details are described as follows.

3.1.1 State. Tiyuntsong's learning agent pushes the input state of time-slot t $s_t = \{T, d, q, r, b, \mathbb{S}, h\}$ into neural network, 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 of past k sequence; r is the video playback time remaining; b is buffer length used by the client; $\mathbb{S}$ is a vector that represents the video sizes of the next video chunk. The last one h is a vector that reflects that extra features of the past, and it is generated by the GAN Enhancement Module (See below).

3.1.2 Action. The action space is discrete, and the output of the policy network is defined as a probability distribution: $f(s_t, a_t)$, meaning the probability of selection action a_t being in state s_t. In this paper, the action a_t is the $n - dims$ vector, which represents the candidate of video bitrate for the next chunk.

3.1.3 Reward. Our reward is defined as a result $r \in \{0, 1\}$ determined by `Rule`. During the training process, we use `Rule` to compute the winning percentage of two agents for each epoch, and the result in which "0" means loss and "1" represents win. Note that `Rule` can be represented as not only a man-made logistic algorithm but also a neural network model generalized by AI. Based on the result, we can estimate the winning percentage w_i for each agent.

3.2 GAN Enhancement Module

In recent work [18, 26], the lifetime of each bitrate decision is modelled as an MDP, meaning that *action* is only related to the status of the target but not relying on the prior states. However, this assumption lacks evidence. [26] only illustrates that throughput factors can be efficiently captured by Hidden-Markov-Model (HMM). Still, in [18], they also consider different numbers of past throughout measurements to represent *state*. In general, previous work leverages a past k steps status observed to approximate the status of the target in MDP, and the limitation of sequence length leads to missing crucial information of the past, such as the maximum and minimum values of the throughput observed.

We, therefore, present *GAN Enhancement Module* to automatically generate the hidden features from the past to break the limitation of sequence length. As is illustrated in Figure 5, the module consists of a generator **G** and a discriminator **D**, where **G** is a function represented by several fully connected layers using *leaky RelU* with parameters θ_g, and **D** is also a function represented by multi-layers with parameters θ_d. For each step t, **G** is used for generating next hidden features h_t according to the state s_{t-1} and hidden feature h_{t-1}, and **D** outputs a single scalar $p_t \in [0, 1)$ to estimate the probability that h_t belongs to the historical winning samples. Motivated by LSGAN [19], we first update **D** by descending its gradient according to $\mathbf{L}_d$ (Eq. 1). We then update **G** by descending its gradient via $\mathbf{L}_g$ (Eq. 2). Here $\mathbf{w}$ means the winning samples which

[2] `https://github.com/hongzimao/pensieve`

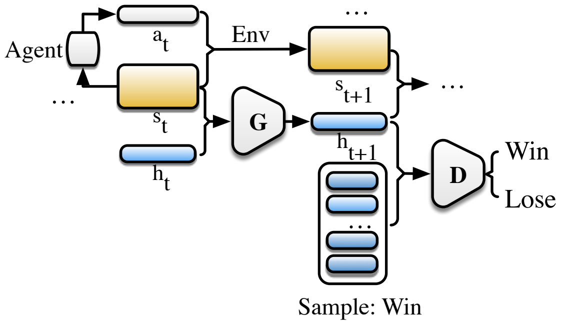

Figure 5: GAN Enhancement Module

are defined as $\mathbf{w} = \{h_0, h_1, \cdots, h_k\}$, where reward $r_k = 1$.

$$\begin{aligned}\min_{\mathbf{D}} \mathbf{L}_d =& \frac{1}{2} E_{x\sim p_w(x)}[(\mathbf{D}(x) - 1)^2] + \\ & \frac{1}{2} E_{t\sim p_g(t)}[(\mathbf{D}(\mathbf{G}(s_{t-1}, h_{t-1})))^2]\end{aligned} \tag{1}$$

$$\min_{\mathbf{G}} \mathbf{L}_g = \frac{1}{2} E_{t\sim p_g(t)}[(\mathbf{D}(\mathbf{G}(s_{t-1}, h_{t-1})) - 1)^2] \tag{2}$$

Algorithm 1 Tiyuntsong's Overall Training Procedure

```
Require: The ABR environment Env to measure total bitrate, total rebuffer time, and total bitrate change with given video samples $\mathbf{V}$ and network traces $\mathbf{T}$; Two agents with GAN Enhance Module $\{\mathbf{A}_0; \mathbf{G}_0, \mathbf{D}_0\}$ and $\{\mathbf{A}_1; \mathbf{G}_1, \mathbf{D}_1\}$; A rule for estimating reward Rule;.
procedure TRAINING(Env, $\mathbf{T}, \mathbf{V}, \mathbf{A}_0, \mathbf{A}_1$)
    Initialize the parameters $\theta$ of $\theta_{\mathbf{A}_0}$ and $\theta_{\mathbf{A}_1}$ with random weights respectively.
    repeat                                        ▷ Epoch ← Epoch + 1
        Sample trace, video from $(\mathbf{T}, \mathbf{V})$;
        for $(t, v) \in$ (trace, video) do
            $(s_0, a_0) \leftarrow$ Env$(\mathbf{A}_0, t, v)$;
            $(s_1, a_1) \leftarrow$ Env$(\mathbf{A}_1, t, v)$;
        end for
        Compute reward $\mathbf{R} \in \{r_0, r_1\} \leftarrow$ Rule$(s_i, a_i)$;
        Estimate winning percentage $\mathbf{w} \in \{w_0, w_1\}$ from $\mathbf{R}$;
        for $i \in \{0, 1\}$ do
            Get winning samples $\mathbf{A}_{i_w}$;
            Update $\mathbf{D}_i$ and $\mathbf{G}_i$ with (1) and (2) using $(s_i, a_i, r_i, \mathbf{A}_{i_w})$;
            Update policy with (4) and (5) using $(s_i, a_i, r_i, w_i)$;
        end for
    until Converged
end procedure
```

3.3 Training Methodology

We now start to discuss how to train Tiyuntsong. In our work, inspired by A3C [20], we use the actor-critic method as the fundamental algorithm of Tiyuntsong. Each agent is composed of a policy network and a value network. The key thought of the policy gradient algorithm is to update the parameter in the direction of increasing the accumulated reward. The gradient of the accumulated reward with respect to policy parameter θ can be written as:

$$\nabla E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = E_{\pi_\theta}[\nabla_\theta \log_{\pi_\theta}(s, a) A^{\pi_\theta}(s, a)] \tag{3}$$

We can use: $E_\theta[\nabla_\theta log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$ as its unbiased form, where $A(s_t, a_t)$ is called the advantage of action a_t in state s_t which satisfies the following equality: $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$, where $V(s_t)$ represents the estimate of the value function of state s_t and $Q(a_t, s_t)$ is the value of taking certain action at in state s_t. Next, we consider to use n-step Q-learning for optimizing the value network. The value network will be updated as:

$$\theta_v \leftarrow \theta_v - \alpha_v \sum_t \nabla_{\theta_v}(r_t + \gamma V(s_{t+1}|\theta_v) - V(s_t|\theta_v))^2. \tag{4}$$

Here $V(s_t|\theta_v)$ is the estimation of $V(s_t)$, the direction of changing parameter θ_v is the negative gradient of it;α_v is the learning rate for the value network. We also add the entropy of policy in the object of policy network, which can effectively discourage converging to sub-optimal policies. So the update of θ will be written as:

$$\theta \leftarrow \theta + \alpha_p \sum_t \nabla_\theta \log_{\pi_\theta}(s_t, a_t) A(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)), \tag{5}$$

where $H(\cdot)$ is the entropy of the policy. After convergence, the value network will be abandoned, and we only use policy network to make decisions; α_p is a learning rate function; β is a hyper-parameter regarded as the weight of exploration; For each epoch $i(i > 0)$, the parameters α_{p_i} and α_{v_i} can be computed by the equalization as follows:

$$(\alpha_{p_i}, \alpha_{v_i}) = \begin{cases} (\alpha_{p_0}, \alpha_{v_0})[(w_i + \epsilon)\log(w_i + \epsilon) + 2.0] & w_i < 0.5 \\ -(\alpha_{p_0}, \alpha_{v_0})(w_i + \epsilon)\log(w_i + \epsilon) & w_i \geq 0.5, \end{cases} \tag{6}$$

in which w_i is the winning percentage for each training epoch i, α_{p_0} and α_{v_0} are initialized hyper-parameters which control the overall learning rate of policy network and value network. Dynamic learning rate can avoid a huge gap between the two agents. See details in Algorithm 1.

3.4 Parallel Training

During the training process, we observe that the training progress is inefficient while using a single process. Inspired by the multi-agent training method [20], we modify Tiyuntsong's training in the single agent as training in multi-agents. Multi-agents training consists of two parts, a central agent and a group of forwarding propagation agents. The forward propagation agents only decide with both policy and critic via state inputs and neural network model received by the central agent for each step; then it sends the n-dim vector containing $\{state, action, reward, gan\}$ to the central agent. The central agent uses the actor-critic algorithm to compute gradient and then updates its neural network model. Finally, the central agent pushes the updated network parameters to each forward propagation agent. Note that this can happen asynchronously among all agents, for instance, there is no locking between agents. By default, Tiyuntsong with multiple training uses 12 forward propagation agents and one central agent;

4 EVALUATION

4.1 Experimental Setup

4.1.1 Datasets. We collect network traces from different public datasets for training and evaluating Tiyuntsong. The detail of our network traces is described as follows:

- Norway [22]: a well-known 3G/HSDPA network trace dataset. We use a subset of 86 traces.
- Synthetic Network Traces [18]: A Markovian-based model which can generate synthetic traces. We create a dataset of over 100 traces which covers a board set of network conditions.
- Belgium [29]: a small 4G dataset which consists of 40 traces.
- FCC [21]: a broadband dataset. In short, we use a subset of 1,000 traces.
- Oboe [2]: a new trace dataset collected from wired, WiFi and cellular network connections. We totally use 429 traces[3] to evaluate Tiyuntsong.

4.1.2 The Design of `Rule`*.* We repeat that `Rule` is allowed to design in many ways including logistic and AI methods. In this experiment, `rule` is presented as a simple logistic algorithm only to test the feasibility of Tiyuntsong. A good ABR algorithm mainly consists of three underlying metrics [24]:

(1) **Bitrate:** To play the video at the highest sustainable quality, such as bitrate and video quality.
(2) **Rebuffering:** To avoid rebuffering events that occur due to the client buffer being empty.
(3) **Smoothness:** Keep the bitrate in little change during the entire session.

Motivated by these features, we then implement a simple logistic rule for evaluation (See in Table 1). where b_i denotes total bitrate, and r_i is total rebuffer time as well as s_i represents total bitrate change for each agent $i \in \{0, 1\}$.

Table 1: The `rule` Used In The Experiment

(a)

`Rule`	$b_0 > b_1$	$b_0 = b_1$	$b_0 < b_1$
$r_0 > r_1$	Table 1(b)	**1**	**1**
$r_0 = r_1$	**0**	Table 1(c)	**1**
$r_0 < r_1$	**0**	**0**	Table 1(b)

(b)

$\frac{r_0}{b_0+\epsilon} > \frac{r_1}{b_1+\epsilon}$	**1**
$\frac{r_0}{b_0+\epsilon} = \frac{r_1}{b_1+\epsilon}$	**0**
$\frac{r_0}{b_0+\epsilon} < \frac{r_1}{b_1+\epsilon}$	Table 1(c)

(c)

$s_0 > s_1$	**1**
$s_0 = s_1$	**0/1**
$s_0 < s_1$	**0**

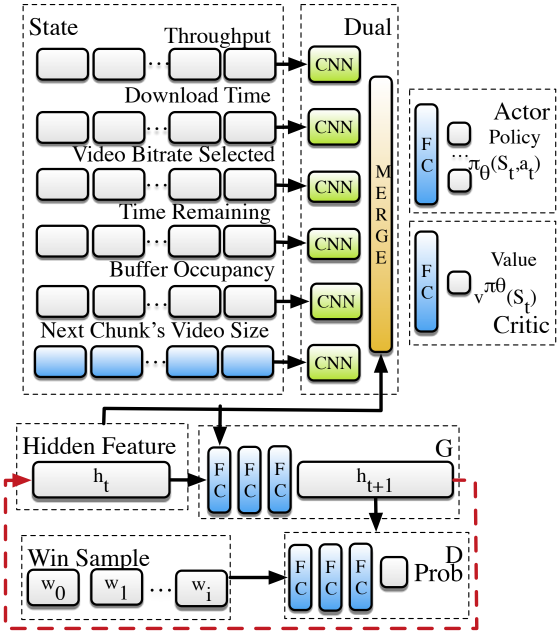

Figure 6: Tiyuntsong's network architecture.

4.1.3 Metrics. We leverage Elo Ratings [8], a traditional method for calculating the relative performance of players in zero-sum games, to evaluate Elo ratings based on winning percentage. We first select several previously proposed approaches and test their performance on the testbed environment respectively. Next, we use `rule` to estimate the winning percentage. Finally, we compute the Elo rating for these approaches. In our work, these scores are defined as *baselines*. For each epoch, the agent compares the result with baselines, and then computes the Elo rating through the winning percentage. In this experiment, we set hyperparameter $K = 10$ and the initialized rating $I = 1000$ of the Elo-rating system.

4.1.4 Testbed Setup. We leverage Sabre [24], a state-of-the-art open-sourced simulation environment for ABR algorithms [4] to precisely emulate the ABR's process in an offline environment. Sabre is a Python tool that can quickly evaluate ABR algorithms in an emulated environment similar to real production players. For each step t, the agent uses the Sabre environment to simulate the entire session with given video descriptions and network traces.

4.2 Architecture and Implementation Details

We use TensorFlow [1] to implement Tiyuntsong[5]. As demonstrated in Figure 6, Tiyuntsong is composed of five neural network architectures as follows.

Dual network: We set past sequence length $k = 10$. Features are extracted from the input state via a feature extraction layer. For each feature in the input state, it's passed through a conv-1d layer with 64 filters and the kernel size of 3×3. Meanwhile, we use

[3] `https://github.com/USC-NSL/Oboe`

[4] `https://github.com/UMass-LIDS/sabre`

[5] [Online]Available: `https://github.com/thu-media/tiyuntsong`

ReLU function as the activation function after each layer. Finally, the feature maps are concatenated as a tensor.

Policy network & Value network: Both policy network and value network are performed behind the Dual network. We use a fully connected layer with 64 neurons and active function *ReLU* to represent them. The output of each network is n-dim vector and a single scalar respectively. In this work, we set $\gamma = 0.6$, $\beta = 0.01$, the learning rate for policy network $\alpha_0 = 10^{-4}$, and the learning rate for value network $\alpha_v = 10^{-3}$. In this experiment, we use Adam optimizer [14] with default parameters to optimize these neural networks.

Generative network & Discriminator network: Like previous work, the generative network and discriminator are composed of fully connected layer FC and batch normalization layer BN. The generative network architecture is described as $\mathtt{FC}^1_{64} \rightarrow \mathtt{BN}^1 \rightarrow \mathtt{FC}^2_{32} \rightarrow \mathtt{BN}^2 \rightarrow \mathtt{FC}^3_{16}$, and the discriminator network is listed as $\mathtt{FC}^1_{64} \rightarrow \mathtt{BN}^1 \rightarrow \mathtt{FC}^2_{32} \rightarrow \mathtt{BN}^2 \rightarrow \mathtt{FC}^3_1$. Meanwhile, we use *Leaky ReLU* as the active function and set learning rate for the generative network and discriminator network $\alpha_G = \alpha_D = 10^{-4}$, hidden feature size $size_{h_t} = 16$. Referring to the recommendations in LSGAN [19], we use RMSProp optimizer [27] to update their gradients.

4.3 Tiyuntsong's Training Time

Tiyuntsong trains itself via endless competition, so the longer Tiyuntsong trains, the better it performs. In this paper, we stop training Tiyuntsong on i7-4790k CPU in 4 cores till its Elo-rating exceeds previous approaches. (Unlike traditional CV work, AI in networking requires a small model which can obtain high performance in low costs, so training on CPU is feasible). The training time lasts about 1.5 days. We observe that Tiyuntsong outperforms previously proposed approaches in 40mins, 2hrs, 13hrs, and 33hrs respectively. We will focus on accelerating the training process as is described in future work.

4.4 Experiments and Results

We evaluate Tiyuntsong under various network conditions and compare it with several state-of-the-art methods. Our results answer the following questions:

① What's the best neural network architecture for Tiyuntsong?

② Does GAN Enhancement Module work? How many improvements does Tiyuntsong obtain through the GAN Enhancement Module?

③ Comparing Tiyuntsong with previously proposed approaches on the same network conditions, does Tiyuntsong stand for the best method?

4.4.1 Tiyuntsong with Different Architectures. In this experiment, we compare the Dual network architecture from Tiyuntsong to the following network architectures which collectively represent the architecture candidates. The network architecture candidates are simply listed as follows:

- **Fully Connected:** $\mathtt{FC}^1_{64} \rightarrow \mathtt{FC}^2_{128} \rightarrow \mathtt{FC}^3_{64}$
- **LSTM:** $\mathtt{LSTM}^1_{64} \rightarrow \mathtt{LSTM}^2_{64} \rightarrow \mathtt{SELF\text{-}ATTENTION}^1_{64}$
- **2D-CNN:** $\mathtt{CONV2D}^1_{64} \rightarrow \mathtt{MAXPOOL}^1_2 \rightarrow \mathtt{CONV2D}^2_{64} \rightarrow \mathtt{MAXPOOL}^2_2 \rightarrow \mathtt{FC}^1_{64}$
- **1D-CNN*:** $\mathtt{CONV1D}^{1\cdots6}_{64} \rightarrow \mathtt{MERGE}^1 \rightarrow \mathtt{FC}^1_{64}$

Arch.	Elo	Timespan(it/s)
FC	1033	**1.28**
LSTM	1057	0.77
2D-CNN	1040	1.16
1D-CNN	**1094**	1.04
Constrained	977	-
Throughput-Rule	1023	-

Table 2: Comparing performance (Elo ratings) of Tiyuntsong with different neural network architectures including Fully Connected, 2D-CNN, 1D-CNN and LSTM. Results are evaluated under same network traces and video description in 50 steps.

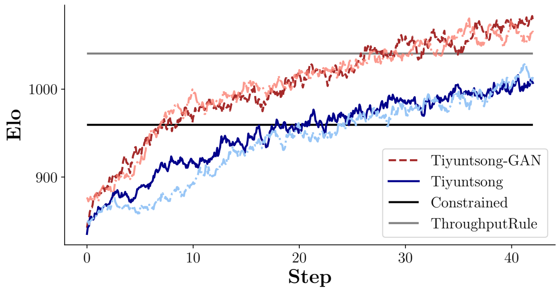

Figure 7: Comparing Tiyuntsong-GAN with Tiyuntsong on the same network traces. Results are evaluated in Elo ratings based on previous approaches as baselines.

We train and test under Sabre environment with the same network traces and video descriptions. In this experiment, we set $\gamma = 0.99$, $\beta = 0.02$, $step = 50$ for only testing their performance instead of convergence. We report the result in Figure 2, where 1D-CNN is the Tiyuntsong's Dual network architecture. The obtained results indicate that 1D-CNN neural network architecture succeeds in improving the Elo ratings, with improvements in average Elo ratings of 37 - 61. We also observe that there is no obvious difference between these architectures in terms of operational efficiency.

4.4.2 Tiyuntsong vs. Tiyuntsong without GAN. In this part, we design an experiment to confirm whether the GAN Enhancement Module is effective or not. We set $step = 50$ and compare Tiyuntsong-GAN with Tiyuntsong without using GAN Enhancement Module on the same network traces, and use two existing approaches: constrained and throughput rule as baselines. The experimental result is illustrated in Figure 7. As expected, we observe that Tiyuntsong-GAN is able to outperform Tiyuntsong, with improvements in average Elo ratings of 13.3% after 50 steps.

4.4.3 Tiyuntsong vs. Existing ABR Approaches. In this experiment, we aim to evaluate the Elo ratings of several existing ABR algorithms including BOLA, DynamicDASH, Throughput-based, Constrained, and Pensieve (QoE-lin). BOLA and DynamicDASH have been implemented in [24], and we use the harmonic mean of past five throughput measured to present the throughput-based

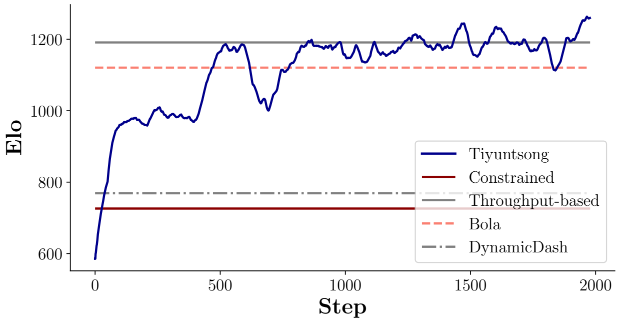

Figure 8: The curve of training Tiyuntsong for 2,000 steps. Elo-ratings are computed from Rule between different ABR algorithms including Constrained, Throughput-Based, Bola and DynamicDASH.

rule. Moreover, we denote the constrained rule as to select the intermediate chunk of the next video chunks and train a model via Pensieve optimized by `QoE-lin` [30]. We train Tiyuntsong about 2,000 epochs on the network traces datasets. We use 80% dataset for training and 20% for validating, and leverage Oboe dataset for testing. Figure 8 shows the performance of training Tiyuntsong for 2,000 steps, we observe that Tiyuntsong performs better than the existing approaches after 1,800 steps. We also report Tiyuntsong's winning percentage and the CDF distribution of three underlying metrics in Figure 9. Compared to DynamicDash, Tiyuntsong improves the average bitrate by 3.19%, decreases the average rebuffer time by 4.92%, and reduces the 95th percentile average bitrate change by 16.47% respectively. As expected, we also observe that Pensieve does reach an overwhelming advantage on the QoE metric but fails to perform well under some underlying metrics such as average bitrate, and it also proves our movitation of this work.

5 RELATED WORK

5.1 ABR Algorithms

Client-based ABR algorithms are mainly organized into four types [3]: throughput-based, buffer-based, mixed and RL-based.

Throughput-based: The development of ABR algorithms begins with the idea of predicting throughput. PANDA [16] predicts the future throughput for eliminating the ON-OFF steady issue. FESTIVE [13] estimates future throughput via the harmonic mean of the throughput measured for the past five (or twenty) chunk downloads. However, due to the lack of throughput estimation method currently, these approaches still result in poor ABR performance.

Buffer-based: Most video client leverages a playback buffer to temporarily store the video content downloaded from the server. Thus, many approaches are designed to select the appropriate high bitrate next video chunk and avoid rebuffering events based on playback buffer size observed. BBA [12] proposes a linear criterion threshold to control the available playback buffer size. BOLA [25] turns the ABR problem into a utility maximization problem and solve it by using the Lyapunov function. However, the buffer-based approach fails to tackle the long-term bandwidth fluctuation problem.

Mixed: Then, mixed approaches, such as MPC [30] and DynamicDASH [24], 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 ([2] even proposes an auto-tuning method) because they rely on parameters that are quite sensitive to network conditions, resulting in the poor performance in unexpected network environments.

RL-based: To address these issues, several attempts [4, 6] have been made to optimize ABR algorithm based on RL method due to the difficulty of tuning mixed approaches for handling different network conditions. Pensieve [18] is a system that uses DRL to select bitrate for future video chunks. D-DASH [9] uses Deep-Q-learning method to perform a comprehensive evaluation based on state-of-the-art algorithms, including both heuristics and learning-based.

5.2 Adversarial Learning

Since GAN first proposed [10], the adversarial discriminative learning method has been widely used in the various fields. The original GAN model is short of the loss function. Thus, many approaches, such as LSGAN [19] and WGAN-gp [11], extend GAN's training methodology with strong theoretical proof. Moreover, adversarial learning has also been applied to extract features. For example, CoGAN [17] uses GAN to solve the domain transfer issue, and ADDA [28] proposes a GAN-based generalized framework for domain adaptation.

6 DISCUSSIONS

6.1 Traditional QoE functions and Rules

In this work, we find that more than one rule can be generalized to represent the same QoE formulation, vice versa. For example, during the design of method, fixed-rules, such as throughput-based and buffer-based, use handcraft features or network presumptions to implement the model without considering how to take advantage of evaluation metrics (QoE formulation). Then, the given QoE formulation is only used to evaluate the performance of each algorithm. Furthermore, mixed-based and RL-based schemes, i.e., MPC [30] and Pensieve [18] adopts the QoE formulation to *guide* its algorithm for achieving higher QoE score. However, recent research [?] exposes that there still exists a plenty of room for improving QoE metrics and many situations (e.g., some network conditions and videos) cannot be evaluated correctly via current QoE metrics due to the lack of features, as the RL-based scheme still tries to optimize the QoE score with the false guidance, finally results in failure of real-world performances. As a result, no matter how precisely and carefully the QoE function tunes, traditional RL-methods cannot exactly provide a result that the users desire.

Intuitively, the critical idea of Rule is to tackle the problem that the reward function fails to depict. For example, ABR tasks and self-driving car tasks. The fundamental factor of Rule is: Given two answers (action) from one questions (state), can *you* figure out which one is better to answer? In this paper, we prove that using self-play reinforcement learning will learn the strategy by itself if *you* can *tell* the agent who is better.

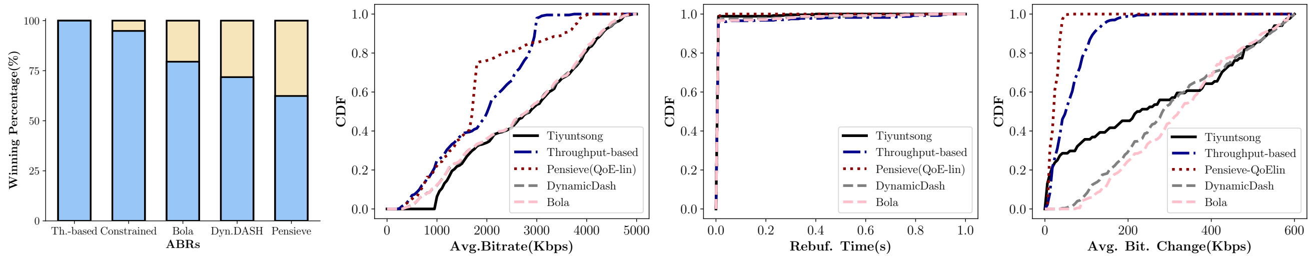

Figure 9: Comparing Tiyuntsong with existing ABR approaches on the same network traces. Results is shown with the winning percentage and the distribution of average bitrate, rebuffering time and average bitrate change for the approaches. Results show that Tiyuntsong wins existing approaches, with the winning percentage of 62% to 100%.

6.2 The Diversity of Network Traces

The real world network is composed of several network conditions such as 3G/HSPDA, 4G, Wired and WiFi. It's obvious that each of them has different features, and we try to train a generalized model which can cover all the network status. Thus, we collect a corpus of network traces by combining several public datasets. Meanwhile, the diversity of the length of network traces is still challenging. On the one hand, each dataset is generated in different durations and granularity. For example, each of FCC dataset we used logs the average throughput about 100 seconds, and at a 5-second granularity (the log is sized 20); Each of synthetic network traces is logged as the average throughput about 2000 seconds. On the other hand, we balance the data distribution by controlling the amount of various network traces in the data pool during the training process. Additional information will be open-sourced later on.

7 CONCLUSION AND FUTURE WORK

We propose Tiyuntsong, self-play RL approach to select bitrates for next video chunk. Unlike previously proposed approaches, Tiyuntsong uses two agents to compete against each other for automatically generating a better ABR algorithm. Experimental results prove that Tiyuntsong has achieved the state-of-the-art ABR algorithm via self-play. Additional research may focus not only to accelerate the training process but also to extend our work to solve the general incomplete information game problem.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 ACM SIGCOMM Conference*. ACM, 44–58.

[3] Abdelhak Bentaleb, Bayan Taani, Ali C Begen, Christian Timmerer, and Roger Zimmermann. 2018. A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP. *IEEE Communications Surveys & Tutorials* (2018).

[4] Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard. 2016. Online learning adaptation strategy for DASH clients. In *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 8.

[5] Cisco. 2017. Cisco Visual Networking Index: Forecast and Methodology, 2016-2021. (2017). https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[6] Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. 2014. Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client. *Connection Science* 26, 1 (2014), 25–43.

[7] Thang Doan, Bogdan Mazoure, and Clare Lyle. 2018. GAN Q-learning. *arXiv preprint arXiv:1805.04874* (2018).

[8] A.E. Elo. 1978. *The rating of chessplayers, past and present.* Arco Pub. https://books.google.com/books?id=8pMnAQAAMAAJ

[9] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella. 2017. D-DASH: A Deep Q-Learning Framework for DASH Video Streaming. *IEEE Transactions on Cognitive Communications and Networking* 3, 4 (Dec 2017), 703–718. https://doi.org/10.1109/TCCN.2017.2755007

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*. 5767–5777.

[12] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[13] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (TON)* 22, 1 (2014), 326–340.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541* (2016).

[16] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.

[17] Ming-Yu Liu and Oncel Tuzel. 2016. Coupled Generative Adversarial Networks. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 469–477. http://papers.nips.cc/paper/6544-coupled-generative-adversarial-networks.pdf

[18] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the 2017 ACM SIGCOMM Conference*. ACM, 197–210.

[19] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. 2017. Least squares generative adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2813–2821.

[20] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[21] Measuring Fixed Broadband Report. 2016. 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].

[22] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 114–118.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR* abs/1712.01815 (2017). arXiv:1712.01815 http://arxiv.org/abs/1712.01815

[24] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From theory to practice: improving bitrate adaptation in the DASH reference player. In *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 123–137.

[25] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *INFOCOM 2016, IEEE*. IEEE, 1–9.

[26] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 272–285.

[27] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012), 26–31.

[28] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. 2017. Adversarial discriminative domain adaptation. In *Computer Vision and Pattern Recognition (CVPR)*, Vol. 1. 4.

[29] Jeroen van der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. 2016. HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks. *IEEE Communications Letters* 20, 11 (2016), 2177–2180.

[30] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*. ACM, 325–338.