DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EEE407 FINAL MSc PROJECT

DASH Video Streaming Using future information

(Final Thesis)

Thesis submitted to the
Xi'an Jiaotong-Liverpool University
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Student Name	:	Tiantian Guo
Student ID	:	1715387
Supervisor	:	Jimin Xiao
Assessor	:	Qiufeng Wang

Abstract

DASH (Dynamic adaptive video streaming over HTTP) has quickly become a hot research topic in the field of streaming media at home and abroad. In this paper, we quantified the client viewing experience into a concrete calculation and discussed three factors that affect quality of experience (QoE), which means a suitable selection of different video quality of proper representation to download. We discussed that the weights of the three factors have different degrees of influence on QoE performance. Buffer reservation and starvation has a great impact on QoE. Using future information is important but more future information could not make wiser decision. Stable bandwidth and long buffer reservation will lead to more quality switching loss.

To compare, we also calculate the final QoE value with the decided quality level for each segment with introducing reinforcement learning algorithm. However, using simple Q-learning algorithm dose not provide better QoE performance. Simple Q-learning is not suitable for quality level decision. In the end, we introduce the Deep Q-learning method and build a simple network. The QoE performance is no less than traditional method. The training video should be longer than the testing video in order to make wiser decisions on quality choosing.

Key words: DASH, Reinforcement learning, QoE, Deep Q-learning

Content

1. Introduction

1.1. Background

In recent years, more and more attention has been paid to multimedia content transmission, especially video streaming services. Streaming media technology uses streaming to transmit audio and video. People don't have to wait until the entire video file is downloaded before playing. They only need to wait for a short start-up time before they can play the video continuously. Because video files are large and subject to network bandwidth conditions, traditional multimedia technologies require a large cache and a long wait for video downloads to complete before the video can be played. In addition, streaming media data is very real-time. The quality of streaming media can be controlled. If the user's bandwidth conditions allow, then higher quality video can be decoded. On the contrary, if the user bandwidth condition is not very good, then the video decoding speed can be slowed down. Quality will also drop.

Reliable video streaming is supported on the Internet that is trying to deliver, and http-based adaptive streaming HAS (HTTP adaptive streaming) become a trend in video business technology. This method uses a simple and convenient HTTP protocol and a common web server [1]. At the same time, in order to make the media data suitable for HTTP transmission, the video data is divided into smaller video fragments and a small media description file is maintained.

1.2. DASH

In order to adapt to the development of the streaming media industry and the requirements of the corresponding industry, the International Motion Picture Experts Group (MPEG) began to collect the draft of the HTTP streaming standard in April 2009 worldwide. Two years later, with the participation of other relevant international standards organizations, such as 3GPP and some other experts, a unified international standard was finally developed, which is now familiar with MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH). Since its publication, MPEG-DASH has quickly become a hot research topic in the field of streaming media at home and abroad.

Essentially, MPEG DASH specifies the metadata and media formats exchanged between the client and the server. In DASH systems, video is encoded with different method, which leads to different bitrates. Each encoded video is then divided into small video segments, and each has duration for a few seconds [2]. Media Presentation Description (MPD) files contain content information such as video profiles, metadata, server IP addresses, and download URLs [3]. The client first requests the MPD file. According to the MPD file requested, it can know the video information corresponding to the link, including the following information:

- Live/on-demand
- The length of each segment to be shard
- The whole length of video
- How many different rates, how many different resolutions
- Video fragment URL address information (which can be inferred by similar naming in turn)

- The time for the live broadcast to begin

In short, it is to store different bitrates and multiple shards with different resolutions and corresponding description file MPD of the same content in advance on the server side. When playing, the client side can choose the most appropriate version according to its own performance and network environment. The DASH system is shown in Figure 1 [4].

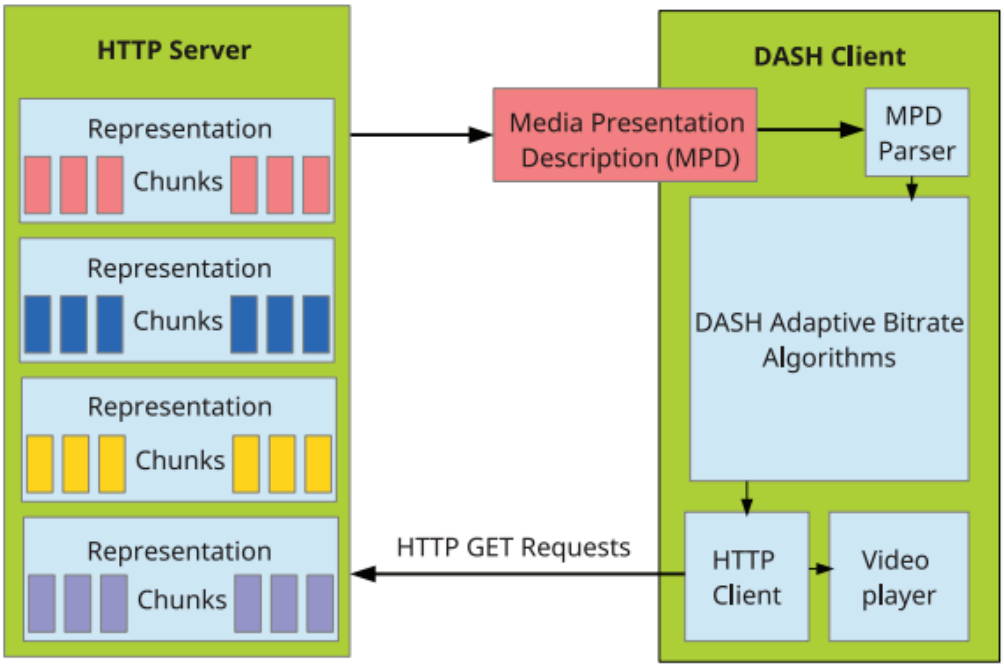


Figure 1: DASH client-server architecture

The DASH standard was developed to address the following issues [5]:

- More efficient use of MPEG media for content distribution via HTTP, adaptive, progressive, download or streaming;
- Support live broadcast services;
- More efficient use of traditional HTTP-based CDN networks, proxy servers or firewalls and other network infrastructure components;

- Support for integration with content protection systems to complete content protection.
- For users, it improves bandwidth utilization, which can improve the viewing experience. It is the most direct benefit.

1.3. Reinforcement Learning

Reinforcement learning (RL) is a machine learning technique where an agent is connected to its environment via perception and action. RL focuses on how agents can take a series of behaviors in the environment to achieve maximum cumulative returns. By RL, an agent should know what state of action should be taken. RL is a study of the mapping from environmental state to action. We call this mapping a strategy. However, an agent has little knowledge about the environment [1].

RL has two characteristics:

- Reinforcement learning is Trail-and-error learning [6]. The agent should constantly interact with the environment and obtain the best strategy through trial and error because there is no direct guidance information.
- Delayed returns. It has less guidance for RL, and returns are often given after the fact (the last state), which leads to the question that how to distribute the reward to the front after getting a positive or negative return status.

Reinforcement learning learns the optimal control strategy through early offline training, and then applies the strategy in real-time adaptive control,

which can improve the flexibility and adaptability of the client's rate decision mechanism [7].

1.3.1. Q-learning

The core of Q-learning is Q-table. The rows and columns of Q-table represent the values of state and action respectively, and the value of Q-table $Q(s,a)$ measures how good the states s are at taking action a at present.

Figure 2: Q-learning architecture [8]

In the process of training, we used Bellman Equation to update Q-table.

$$Q(s,a) = r + \gamma * max_{\tilde{a}} Q(\tilde{s},\tilde{a}) \quad (1)$$

The algorithm of Q-learning is shown in Figure 3 [9].

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 3: Q-learning algorithm

1.3.2. Deep Q-learning

There is a problem with Q-table. There may be an infinite number of states in the real situation, so Q-table will be infinite. The solution to this problem is to implement Q-table through neural network. Input state and output Q-values of different actions. It's very easy to express Q-value in terms of neural networks, so Q-value becomes Q- Network.

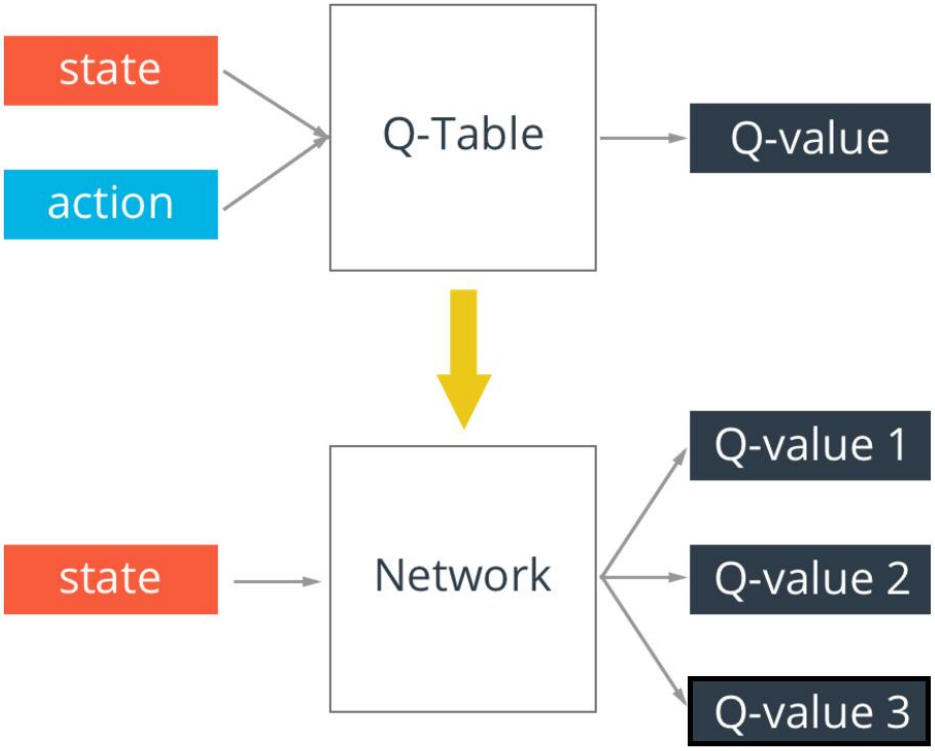


Figure 4: The architecture of Q-learning and DQN [8]

The algorithm of DQN is shown in Figure 5 [10].

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 5: DQN algorithm

2. Literature Review

2.1. HTTP adaptive streaming (HAS)

HTTP adaptive streaming technology automatically selects the video resource with the most appropriate code rate for download by sensing the network, memory, CPU and other conditions of the current host. Throughput-based adaptive algorithm adjusts video bitrate based on estimated throughput, which can be divided into two categories: real-time throughput and smooth throughput [11]. Chen et al. [12] proposed a new adaptive bitrate control algorithm, which takes video resource buffer and network throughput as the main parameters of hybrid control. In the video resource cache control mechanism, the total amount of video data is kept within the equilibrium range by adjusting the time taken for a single video slice to enter the cache area. In the network throughput control mechanism, according to the current network situation, the conservative strategy is used to improve video bit rate, and the reduction of bit rate is determined by using logistic equation.

Due to short-term and long-term fluctuation of TCP data transmission, the bitrate adaptive algorithm that determines video segment based on network throughput may lead to buffer overflow and frequent switching of video bitrate. The buffer-based adaptive algorithm [13] [14] divides the client buffer into multiple wide ranges and selects different bitrate switching strategies according to the state of the buffer region. Paper [15] proposed a hybrid bitrate adaptive algorithm (CBB) that takes network bandwidth and cache state into account. The adaptive detection method is used to predict

the network bandwidth as close as possible to the real-time link bandwidth. The smoothness of the design varies with the cache state because the model dynamically changes the bandwidth smoothness.

The hybrid control's adaptive mechanism takes into account multiple parameters (such as network bandwidth, cache, etc.) to accurately reflect current network conditions and client conditions. The PANDA algorithm proposed by Li et al [16] uses the principle of “probe and adaptive” to detect the real-time bandwidth, and the video code rate is determined based on the “dead-zone” quantizer. The algorithm can maintain the buffer area data amount in a more balanced range as much as possible, and maintain a high average video quality. Hesse et al. [17] proposed a BS algorithm based on media data selection and recovery mechanism during streaming sessions.

2.2. Quality of Experience (QoE)

Paper [18] proposed a concept of QoE to measure the users’ experience and the relationship with quality of service (QoS). They model QoE as a multidimensional structure of user perception and behavior. Ricky et al. [19] provide a similar structure but not base on spatial quality but a structure for HAS. The relationship between QoS and QoE are shown in Figure 6.

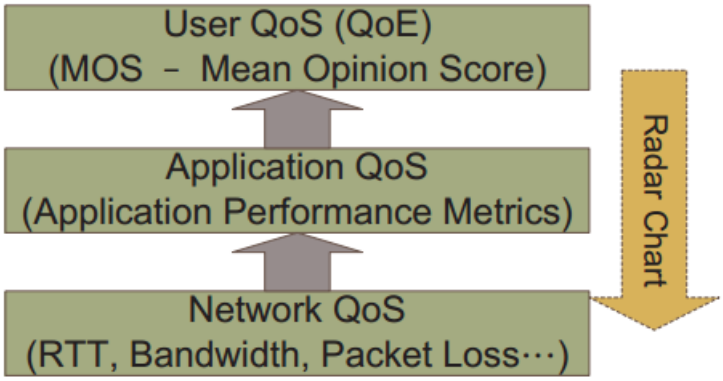


Figure 6: The relationship between QoS and QoE

QoE refers to the user's comprehensive and subjective experience of the quality and performance (including effectiveness and availability) of the device, network, system and application or business, which means it is defined by the comfort of the business application. The QoE rating operator can optimize the network by making a comprehensive assessment of video's business quality and performance.

As introduced above, in order to maximize the user QoE, many studies in this area have attempted to find the best adaptation strategy. Therefore, adaptive algorithms are proposed to monitor the current state of the network, as well as video bitrate and buffer state. Based on this monitoring data, they decide which quality level to request next in order to minimize stagnation [20]. They provided a subjective user study on QoE objectives and tried to optimize the quality adaptation for singer user, which contains the initial delay, possible maximum quality and less quality switching. For multiple users, they also discussed the influence of quality switching.

Chang et al. [21] apply the QoE objectives for users on DASH to evaluate the quality adaptation algorithm. They build a QDASH system, including two parts; one is QDASH-abw, which will evaluate the available bandwidth, the other is QDASH-qoe, which will decide the quality level. The QDASH system provides an QoE-based quality switching algorithm by experiment how video quality adaptation will effect the QoE value.

2.3. Reinforcement learning

DQN uses the Experience Replay Experience pool based on the basic Deep Q-learning algorithm. By storing the training data and then sampling randomly, the correlation of data samples is reduced. The performance is improved [10]. Next, Nature DQN made an improvement by adding Target Q network. That is, we use a specific target Q network to calculate the target Q value, instead of directly using the pre-updated Q network. The goal is to reduce the relevance of the target calculation to the current value.

$$I = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^{-}) - Q(s, a, \mathbf{w}) \right)^2$$

Figure 7: Loss function for Nature DQN [22]

The network calculates the target Q-value uses w^{-}, not w. That is to say, the target Q network of DQN of the original NIPS version changes dynamically with the update of the Q-network, which is not conducive to the calculation of the target Q-value, resulting in a greater correlation between the target Q value and the current Q value. Therefore, a single

target Q network is proposed. The parameters of the target Q network come from the Q-network and are just delayed to update. In other words, after training for a period of time, the parameter value of the current Q network is copied to the target Q-network. The flowchart of Nature DQN is shown in Figure 8.

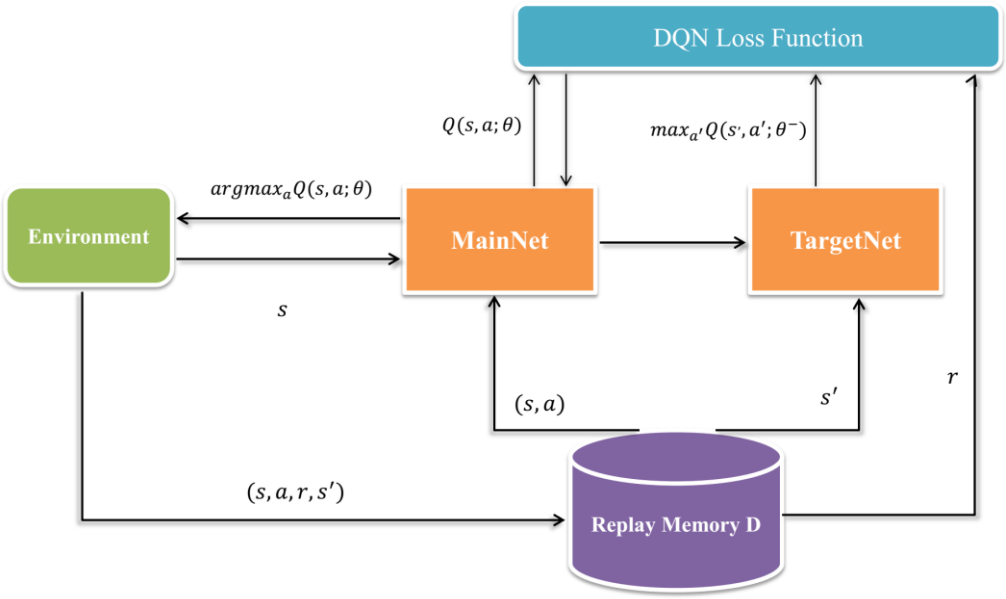


Figure 8: Flowchart for Nature DQN

2.4. DASH and Q-learning

Machine learning method can enhance the flexibility and adaptability of the client rate decision mechanism. Reinforcement learning learns the optimal control strategy through early offline training, and then applies the strategy to real-time adaptive control, which can improve the client's bitrate decision mechanism. Applying reinforcement learning to the quality

selection enables the client to dynamically learn the best behavior in the current network environment.

The introduction of RL methods can enhance the flexibility and adaptability of the client's rate decision mechanism. The literature [23] uses the Markov decision process (MDP) to drive the rate adjustment strategy, considering the key factors affecting the user QoE, and maximizes the quality of the video stream. Applying RL to the heuristics of HAS quality selection enables HAS clients to dynamically learn the best behavior in the current network environment. Istepanian et al. [24] applied the Reinforcement learning method to the field of multimedia transmission to deal with the rate control problem when wireless video transmission of medical video images for the first time.

Claeys et al. [1] designed a HAS client self-learning intelligent system, which divided the environment model into 25 million states. Later, they streamlined the elements of the environmental state, and redesigned the reward function to obtain a better rate decision method [25]. The model proposed by Martin et al. [26] can reasonably balance the complexity of the model with the self-learning ability of the dynamic system, and select the appropriate number of parameters to construct the environmental state according to the user's QoE-related factors (the requested qualities, the quality switches , the freeze) defines the reward.

3. Methodology

3.1. Data preparing

For our work, we choose 3 different lengths of test videos: “big buck bunny.mp4”[27], “bear.mp4”[28] and “test.mp4”. All of them were encoded with four different quality levels to simulate different VBR video versions

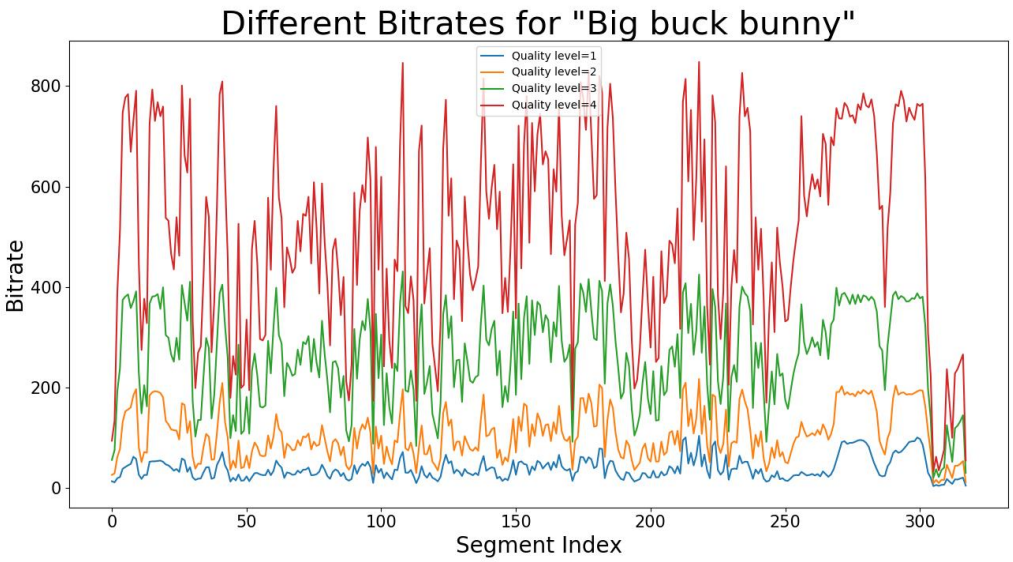
3.1.1. Bitrates

Different videos have different frame rates and resolutions. Each test video was encoded with resolutions as 320x180, 640x360, 1280x720 and 1920x1080, and the corresponding bitrates are set as 512, 1024, 2048 and 4096. Each segment was divided with duration $\tau = 2s$. The video encoder is based on [29] and [27]. Every video’s frame rate is calculated with python code shown in Appendix 1 ‘fps,py’. For “big buck bunny.mp4”, its frame rate is 30 fps and has total 318 segments. For “bear.mp4”, its frame rate is 24 fps and has total 182 segments. For “test.mp4”, its frame rate is 30 fps and has total 130 segments.

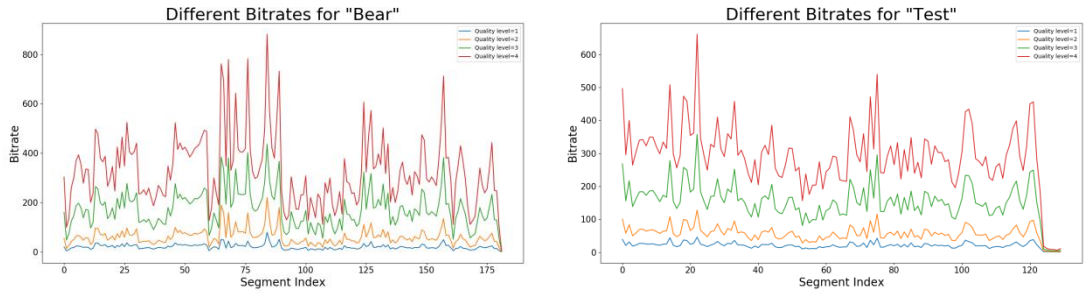
To calculate the bitrate, we assume that in one segment, the bitrate is the same. Bitrate of each segment of each video is calculated with Appendix 2 ‘bitrate.py’. We get the size of each segment of each quality level and divide them by preset duration. The average bitrates of each quality level of each video are shown in Table 1. The bitrates versus segments are shown in Figure 9.

Table 1: The average bitrates for each quality level of different video

Test Video	Quality Level	Average Bitrate (kbps)
Bigbuckbunny	1	38.11423
	2	105.612
	3	262.7073
	4	511.2412
Bear	1	21.3112
	2	61.3485
	3	173.0041
	4	329.2732
Test	1	20.9065
	2	56.4621
	3	154.174
	4	292.2326



(a) Different bitrates for ‘big buck bunny.mp4’;



(b) Different bitrates for ‘bear.mp4’; (c) Different bitrates for ‘test.mp4’

Figure 9: Different bitrates for each video

From Figure 9, we can find that these curves from any of the test videos are similar in shape because the same information in each segment.

3.1.2. Bandwidth

We apply Markov channel model to bandwidth prediction. There are total L states of bandwidth level in Markov chain. The transition probabilities of each bandwidth level B_i change to another level B_j is $P_{i,j}$. Figure 10 shows a 5-state Markov chain model and some sample probabilities are shown.

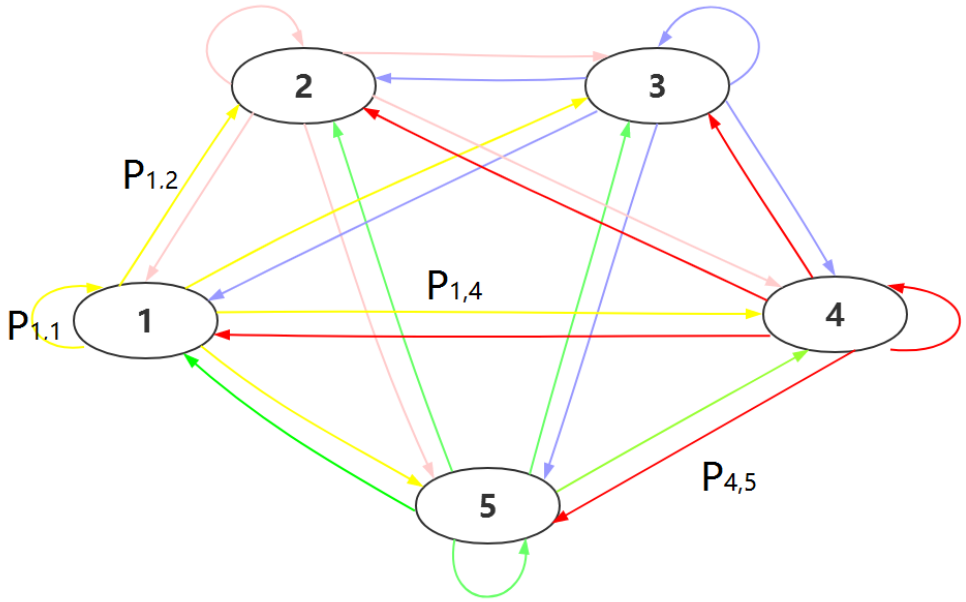


Figure 10: 5-state Markov chain model

The transition matrix is shown in equation (2).

$$A = \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,L} \\ P_{2,1} & P_{2,2} & & P_{2,L} \\ \vdots & & \ddots & \vdots \\ P_{L,1} & P_{L,2} & \cdots & P_{L,L} \end{bmatrix} \quad (2)$$

In general, one bandwidth level has very low possibilities to change to a non-adjacent level. Thus, equation (3) can be considered.

$$P_{i,j} = 0, \text{ if } |i - j| > 1 \quad (3)$$

Assume the $(i-1)^{th}$ segment was downloaded under the bandwidth $b_{i-1} = B_j$, $b_i{}'$, which the estimated bandwidth next decision

time, could be all the L bandwidth levels. Different from [30], we do not consider that bandwidth could not change to non-adjacent level because the probability is set to zero. For example, in Table 2, $\theta_3 = \{B_j, B_1, B_1, \dots, B_1, B_3\}$, but $P_{1,3} = 0$. So $P(\theta_3) = 0$.

Table 2: Probabilities of different bandwidth patterns

Bandwidth patterns	Probability of each bandwidth patterns
$\theta_1 = \{B_j, B_1, B_1, \dots, B_1, B_1\}$	$P(\theta_1) = P_{b_{i-1},1} * P_{1,1} * \dots * P_{1,1} * P_{1,1}$
$\theta_2 = \{B_j, B_1, B_1, \dots, B_1, B_2\}$	$P(\theta_2) = P_{b_{i-1},1} * P_{1,1} * \dots * P_{1,1} * P_{1,2}$
$\theta_3 = \{B_j, B_1, B_1, \dots, B_1, B_3\}$	$P(\theta_3) = P_{b_{i-1},1} * P_{1,1} * \dots * P_{1,1} * P_{1,3}$
...	...
$\theta_{L^{l+1}} = \{B_j, B_L, B_L, \dots, B_L, B_L\}$	$P(\theta_{L^{l+1}}) = P_{b_{i-1},L} * P_{L,L} * \dots * P_{L,L} * P_{L,L}$

As list in Table 2, the probability of a sequence of bandwidth level $\theta_k = \{b_i', b_{i+1}', \dots, b_{i+l}'\}$ is shown in equation (4).

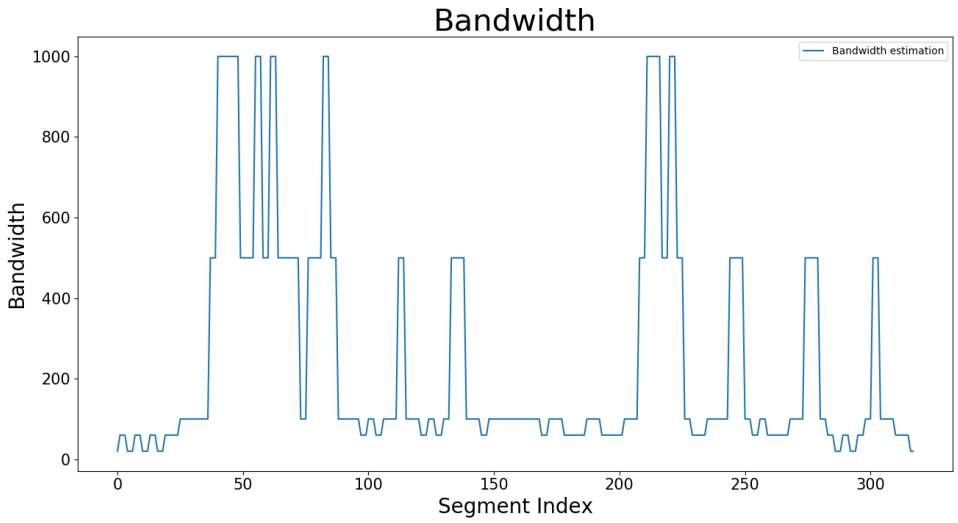
$$P(\theta_k) = P_{b_{i-1},b_i'} * \prod_{j=0}^{l-1} P_{b_{i+j}',b_{i+j+1}'} \quad (4)$$

The Markov chain model provides all possible bandwidth patterns $\{\theta_1, \theta_2, \dots, \theta_{L^{l+1}}\}$ with the corresponding probabilities.

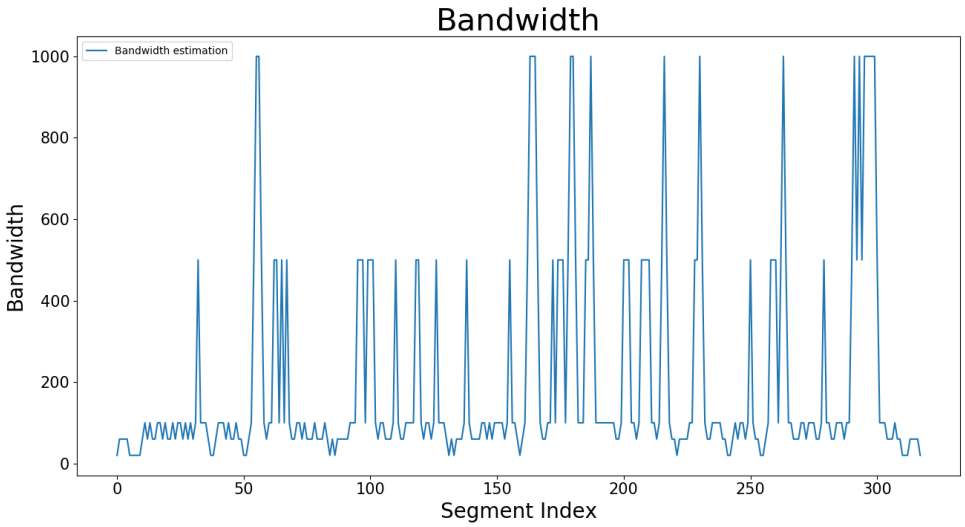
In our work, we set 5 levels of bandwidth, which is $\{20, 60, 100, 500, 1000\}$. To simulate the real network, we set the lowest level of bandwidth (20) is smaller than the lowest average bitrate (21). The transition matrix is shown in equation (5).

$$A = \begin{bmatrix} 0.4 & 0.6 & 0 & 0 & 0 \\ 0.2 & 0.4 & 0.4 & 0 & 0 \\ 0 & 0.4 & 0.4 & 0.2 & 0 \\ 0 & 0 & 0.5 & 0.3 & 0.2 \\ 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix} \quad (5)$$

In our work, we consider two situations: one is that without future information, we only predict the next bandwidth level. The other is we need to predict future l segments' bandwidth value. To simulate the real bandwidth, we assume two situations about the bandwidth estimation: one is every future segments have a different bandwidth value; the code is shown in Appendix 3. The other is assuming the future l segments have the same bandwidth value. The python code is shown in Appendix 4 'bw.py' and the python code is shown in Appendix 3 'bandwidth.py'. Two examples of bandwidth estimation are shown in Figure 11.



(a) Future *l* segments have same bandwidth level



(b) Future *l* segments have different bandwidth level

Figure 11: Different future *l* segments bandwidth

3.2. Baseline

QoE (Quality of Experience) refers to the user's comprehensive and subjective experience of the quality and performance (including effectiveness and availability) of the device, network, system and application or business, which means it is defined by the comfort of the business application. The QoE rating operator can optimize the network by making a comprehensive assessment of video's business quality and performance. Our work is to choose the proper video quality level to get the highest QoE value under the condition of the network, which is affected by the bandwidth and the buffer size, so that client can have better video playing performance.

3.2.1. QoE

First, we need to insert the method to calculate the QoE arithmetically. According to [4], the influence factors of QoE can be divided by two categories: perceptual and technical factors. As in [30], QoE is mainly influenced by three related aspects: the mean of quality level of video, the loss caused by quality switching during video playing and the overflow risk of cache.

As we can see in Table 3, it is assumed that there are total N segments in a video sequence. Each of them has duration for τ seconds. In this video sequence, we assume that it is downloaded with a quality sequence $\varphi = \{q_1, q_2, \dots, q_N\}$ with the network bandwidth sequence $\theta = \{b_1, b_2, \dots, b_N\}$.

Table 3: The definition for each symbol

Symbol	Definition
M	Number of quality level
N	Number of segments
τ	Duration of each segment
$t_i (1 \le i \le N)$	Index of decision point
$q_i (1 \le i \le N)$	Index of possible quality level at decision point t_i, $1 \le q_i \le M$
$b_i (1 \le i \le N)$	Index of possible bandwidth level at decision point t_i, $1 \le b_i \le L$
θ	A sequence of bandwidth levels, i.e. $\theta = \{b_1, b_2, \dots, b_N\}$
φ	A sequence of quality levels, i.e. $\varphi = \{q_1, q_2, \dots, q_N\}$

According to three factors that affect the QoE value, the final QoE formulation is shown in equation (6). $w1$、$w2$ and λ are the weights of each factors.

$$QoE(\theta, \varphi) = E(\varphi) - w1 * V(\varphi) - w2 * P^S(\theta, \varphi) \quad (6)$$

The average quality level $E(\varphi)$ is defined as the average of all quality levels.

$$E(\varphi) = \frac{1}{N}\sum_{i=1}^{N} q_i \tag{7}$$

The loss caused by quality switching $V(\varphi)$during video playing can be calculated by the average difference of quality level between two adjacent segments.

$$V(\varphi) = \frac{1}{N-1}\sum_{i=1}^{N-1} |q_{i+1} - q_i| \tag{8}$$

As we can see in Table 4, the overflow risk of cache is denoted by the total starvation time $T^s(\theta, \varphi)$ over total playout time $T^t(\theta, \varphi)$ as shown in equation (9).

Table 4: The definition for each symbol

Symbol	Definition
$T_i (1 \le i \le N)$	Length of buffer reservation at decision point t_i
$T_i^s (1 \le i \le N)$	Duration of starvation
$T^s(\theta, \varphi)$	Total duration of starvation for one sequence
$T^t(\theta, \varphi)$	Total playout time for one sequence

$$P^s(\theta, \varphi) = \frac{T^s(\theta,\varphi)}{T^t(\theta,\varphi)} \tag{9}$$

Their relationship is shown in equation (10).

$$T^t(\theta, \varphi) = N * \tau + T^s(\theta, \varphi) \tag{10}$$

Where

$$T^s(\theta,\varphi) = \sum_{i=1}^{N} T_i^s \tag{11}$$

Duration of starvation time T_i^s exists when the size of downloaded segment is lager than the length of buffer reservation before T_{i-1}. The size of downloaded segment can be denoted as the corresponding bitrate r_{i,q_i} with quality level q_i and decision time t_i multiply duration for one segment, and the playing time is the size of downloaded segment divided by the bandwidth B_{b_i}. Of course, duration of starvation time could not be negative.

$$T_i^s = \max(\frac{r_{i,q_i} * \tau}{B_{b_i}} - T_{i-1}, 0) \tag{12}$$

Next buffer reservation T_{i+1} can be denoted by the buffer reservation before T_i minuses the time of playing and pluses the duration of each segment. Buffer reservation could not be negative also.

$$T_{i+1} = \max(T_i - \frac{r_{i,q_i} * \tau}{B_{b_i}} + \tau, 0) \tag{13}$$

Now we need to consider the three weights in equation (6). For $w1$, we consider three extreme cases. For simplicity, we ignore the buffer change and set $QoE(\theta,\varphi) = E(\varphi) - w1 * V(\varphi)$. In case 1, there are $\frac{N}{2}$ segments downloaded with quality Q_{max} and the others are downloaded with quality Q_{min}. So QoE is shown in equation (14). In case 2, all the segments are downloaded with quality Q_{max}. So QoE is shown in equation

(15). In case 3, all the segments are downloaded with quality Q_{min}. So QoE is shown in equation (16).

$$case1: QoE = \frac{1}{2} * (Q_{max} + Q_{min}) - w1 * (Q_{max} - Q_{min}) \quad (14)$$

$$case2: QoE = Q_{max} \quad (15)$$

$$case3: \ QoE = Q_{min} \quad (16)$$

To the clients, case 2 has the maximum QoE and case 3 has the minimum QoE. So $QoE_{case3} \leq QoE_{case1} \leq QoE_{case2}$. Also, $w1$ should be a positive value because it is the weight of the loss caused by quality switching. Then, we can get the range of $w1$, which is shown in equation (17).

$$0 \leq w1 \leq \frac{1}{2} \quad (17)$$

$w2$ is related to subjective experience. According to [30], 10% of starvation ratio is equivalent to about two levels drop in quality level, so $w2$ will change according to the value of $P^s(\theta, \varphi)$. In real world, the value of $w1$ and $w2$ could reflect people's preferences. If people prefer fluent quality switching, $w1$ may be set higher so that the value of QoE would be smaller to represent that client is not satisfied. If people prefer higher quality level than fluent playing, $w2$ could be set smaller to reduce the influence of buffer change.

The python code is shown in Appendix 5 'test.py'. In order to achieve highest QoE value, we use every possible current quality level and the predicted bandwidth level to calculate the QoE values and find the maximum one,. Then we will choose the quality level corresponding to the

maximum QoE value. The buffer reservation and starvation ratio are also different in each situation.

3.2.2. Future information

To consider the future information, we need to predict bandwidth of the future l segments. As discussed before, we provided two situations for bandwidth estimation. For quality level, we need to consider all possible quality level for current segment and future l segments. For example, if $l = 2$ and there are total 4 quality levels, we need to consider 3 segments. Each of them has 4 possibilities, so the total possibilities is $4^3 = 64$. The results are shown in Figure 12. In python, we use 'itertools.product' to product the possibilities.

64x3 double

	1	2	3	4
1	1	1	1	
2	1	1	2	
3	1	1	3	
4	1	1	4	
5	1	2	1	
6	1	2	2	
7	1	2	3	
8	1	2	4	
9	1	3	1	
10	1	3	2	
11	1	3	3	
12	1	3	4	
13	1	4	1	
14	1	4	2	
15	1	4	3	
16	1	4	4	
17	2	1	1	
18	2	1	2	
19	2	1	3	
20	2	1	4	

Figure 12: The possible predicted bandwidth

To calculate the QoE with future information, we use two algorithms. One is traditional arithmetic, and the other is based on Q-learning. Q-learning algorithm will introduce next part. For traditional arithmetic, we first use method as discussed before. Every decision point we consider each possibility quality level of current segment and future l segments comes with quality level of segments in the past to calculate QoE value. Then we follow the method in the paper [30], introducing internal QoE.

If we consider the bandwidth level of future l segments is the same as the bandwidth level of the current segment, Internal QoE will include the probability of the bandwidth levels of current segment and future l segments. According to equation (3), we will choose the highest probability. Python code is shown in Appendix 6 'base.py'.

However, because we optimize internal QoE metric to approximate optimal result of final QoE metric, it is important to consider future buffer reservation impacts into the internal QoE assessment. Thus, the length change of the buffer reservation caused by the current decision, called the "buffer change", is considered in the internal QoE metric. Λ is the weight of buffer change in internal QoE metric. The value of λ will influence the importance of buffer change.

The internal QoE is calculated as equation (18).

$$QoE^{inter}(\theta,\varphi) = E(\varphi) - w1 * V(\varphi) - w2 * P^{S}(\theta,\varphi) + \lambda * \Delta T_i(\theta,\varphi) \quad (18)$$

For internal QoE, buffer change could be denoted as equation (19). $T_{i+l}(\theta, \varphi)$ is the estimated length of buffer reservation.

$$\Delta T_i(\theta, \varphi) = \frac{T_{i+l}(\theta,\varphi) - T_{i-1}}{l+1} \quad (19)$$

The Markov channel model provides all possible bandwidth patterns $\{\theta_1, \theta_2, \dots, \theta_{L^{l+1}}\}$ with the corresponding probabilities. Our method is based on internal QoE. We calculate every possible $QoE^{inter}(\theta_k, \varphi_j)$, and multiply with possibilities for every possible φ_j. We find the most possible bandwidth level for each decision point according to the maximum probability. Then we got the sequence of bandwidth level to calculate the QoE.

$$QoE^{inter}(\varphi_j) = \sum_{k=1}^{L^{l+1}} QoE^{inter}(\theta_k, \varphi_j) * P(\theta_k) \quad (20)$$

After we find the max $QoE^{inter}(\varphi_j)$, shown in equation (18), we get the request quality level $\varphi = \{q_1, q_2, \dots, q_N\}$ and bandwidth level $\theta = \{b_1, b_2, \dots, b_N\}$.

$$\max QoE^{inter}(\varphi_j),\ s.t. j \epsilon \{1,2, \dots, M^{l+1}\} \quad (21)$$

That is:

$$QoE^{inter}(\theta_k, \varphi_j) = E(\varphi_j) - w1 * V(\varphi_j) - w2 * P^S(\theta_k, \varphi_j) + \lambda * \Delta T_i(\theta_k, \varphi_j) \quad (22)$$

3.2.3. Q-learning

The traditional method need to calculate the instant QoE value to make decision. In our work, the communication problem between server and client in HAS system is abstracted into dynamic system optimization control problem. We introduce other method to make the decision of quality level for each segment. To compare, we also calculate the final QoE value with the decided quality level for each segment. Q-learning algorithm is a model-independent reinforcement learning algorithm [23], which obtains the knowledge of return expectation and environmental state transition through interaction with the environment, and is used to solve the maximum return and optimal strategy of Markov decision process [7]. In the learning process, Q value represents the knowledge acquired by the Agent. Compromise between exploration and utilization is the most challenging task in Q-learning.

First, the environment state is modeled and the HAS client behavior is defined. We need to consider future *l* segments to estimate QoE value.

1) Environmental status. The interaction behavior between Agent and network environment is modeled according to the bandwidth perceived by the client. It was determined that the environmental state was defined as the following information about the environment at each stage k: $s_k = \{bw'_k\}$, where $bw'_k = \{b_i', b_{i+1}', \dots, b_{i+l}'\}$.

2) Client behavior. By perceiving the current network state, the next state to be transformed is selected from the state reachable set of states, and

each choice is treated as an action. All possible quality levels $a_j = \{q_i', q_{i+1}', \dots, q_{i+l}'\},\ s.t. j \epsilon \{1,2, \dots, M^{l+1}\}$

The buffer reservation will rely on the state and the choose action.

The reward function is the basic wizard for agents to learn the optimal strategy. In video streaming, the return function should be reflected as a measure of the user QoE. According to equation (6):

$$r = QoE^{inter}(s_k, a_j) \ = \ E(a_j) - w1 * \ V(a_j) - w2 * P^S(s_k, a_j) \qquad (23)$$

The total return function will drive the exploration of agents to approach the behavior of higher quality level, less quality switching and more balanced buffer data quantity.

In terms of adopting the exploration strategy, this report first randomly selects the next behavior, traversal all behaviors in the behavior set a_j, and finally performs a specific behavior to obtain the immediate maximum Q value. The idea of Q-learning algorithm proposed is shown in Algorithm 1.

Algorithm 1: Q-learning

Input: the learning rate α, the reward function r, Environment state s, client action a, the contribution of the immediate and future rewards γ,

1) Initialize $Q - table$
2) Select the starting state
3) Select a possible action under the current state
4) Switch to the next state

5) Repeat step 3)
6) Obtaining the return value r. Use Bellman Equation $Q(s,a) = Q(s,a) + \alpha * [r + \gamma * max_{\tilde{a}} Q(\tilde{s},\tilde{a}) - Q(s,a)]$ to update $Q - table$
7) Use the next state as the current state
8) Iterations

Different from other Q-learning, our next state is only base on the current state. As introduced before, next state is predicted from the bandwidth of the last segment in the current state. After enough cycles and iterations, we obtained a trained $Q - table$. When calculating the actual QoE value for comparison, we selected the behavior with the largest value in the $Q - table$ as the quality chosen for these segments by judging what corresponding state the current and predicted bandwidth reflect. The python code is shown in Appendix 7 'qlearning.py'.

However, different from general Q-learning application, our reward function is not only base on the state it stayed and the action it chosen, but also affected by the different bitrates for each segment because we focus on VBR videos and the buffer starvation is calculated by the bitrates and the bandwidth. To solve the problem, we introduce a complex Q-learning algorithm, which has multiple $Q - table$. Since we have total N segments, we will set $N/(l+1)$ $Q - table$s for different bitrates. Each of them will train for enough iteration. The algorithm is update as follows.

Algorithm 2: Multiple Q-learning

Input: the learning rate α, the reward function r, Environment state s, client action a, the contribution of the immediate and future rewards γ,

1) Initialize each $Q-table$
2) Identify the segment index
3) Select the starting state
4) Select a possible action under the current state
5) Switch to the next state
6) Repeat step 3)
7) Obtaining the return value r base on the corresponding bitrates and the bandwidth relate to the segment index. Use Bellman Equation $Q(s,a) = Q(s,a) + \alpha * [r + \gamma * max_{\tilde{a}} Q(\tilde{s},\tilde{a}) - Q(s,a)]$ to update the corresponding $Q-table$ according to the segment index
8) Use the next state as the current state
9) Iteration for enough times
10) Change to the next segment index and repeat from step 3) to step 9)

After finish updating all the $Q-table$, for each decision point, we selected the behavior with the largest value in the corresponding $Q-table$ with the different bitrates. The qualities for current segment and future l segments are chosen by judging what corresponding states the current and predicted bandwidths reflect. The python code is shown in Appendix 8 ‘qlearning2.py’.

3.2.4. Deep Q-learning

In the analysis above, we used a table to represent $Q(s,a)$, but this is almost impossible on many practical problems because there are too many states. In before work, the states are set as the bandwidth of current segments and future *l* segments, but it is an easy way to consider the states.

In real problems, the bandwidth, the bitrates and the buffer change will also represent a new state. The buffer starvation is changing all the time, which affect the reward function. Using tables can not save them. In Deep Q-learning, we can consider more complex states.

There is a concept in Reinforcement learning, called Value Function Approximation. That is:

$$Q(s,a) \approx f(s,a,w) \tag{24}$$

In the training part, we choose the bitrates value of one of the videos. Similar to Figure, we set two simple full connected layers for our network. The input for the network is the bandwidth and the bitrates for each decision point. The input of the first layer is 5, which means four bitrates for different quality level and one bandwidth value. The output of the second layer is 4, which means the Q value for four different quality levels. The information in the states is the bitrates, future segments number and decision point. The reward is set as the difference between the bandwidth and the four bitrates. If the bandwidth is smaller than the bitrates, we set the reward as a higher number.

In this network, our target Q value could be represent by equation (25)

$$Q(s,a) = \alpha * [r + \gamma * Q(\tilde{s},\tilde{a}) - Q(s,a)] \tag{25}$$

The training of neural network is an optimization problem. It optimizes a loss function, that is, the deviation between the label and the network output. The goal is to minimize the loss function. For loss function, we use

Mean Square Error (MSE) loss to measure the loss between Q value and the target Q value. For optimizer, we introduce Adam algorithm. Adam (Adaptive Moment Estimation) is essentially an RMSprop with momentum items [31]. It uses the first-order Moment Estimation and second-order Moment Estimation of the gradient to dynamically adjust the learning rate of each parameter. Its advantage is that after bias correction, the learning rate of each iteration has a certain range, so that the parameters are relatively stable.

In the testing part, we choose the bitrates of another video. The bandwidth prediction is also use the Markov channel model. To compare the QoE value, after finishing the testing, we use the quality chosen and the bandwidth to calculate QoE value as our baseline. The code is shown in Appendix 9 ‘Q.py’, which is from my partner.

4. Results

4.1. Preliminary baseline results

In the first, we need to conform the value of $w1$ and $w2$ We first try to simulate bandwidth chain. One example bandwidth chain is shown in Figure 13. As discussed before, one bandwidth level has very low possibilities to change to a non-adjacent level, so we could not just randomly choose bandwidth levels to simulate the network in the client. We need to choose neighboring bandwidth levels. Python code is shown in Appendix 1.

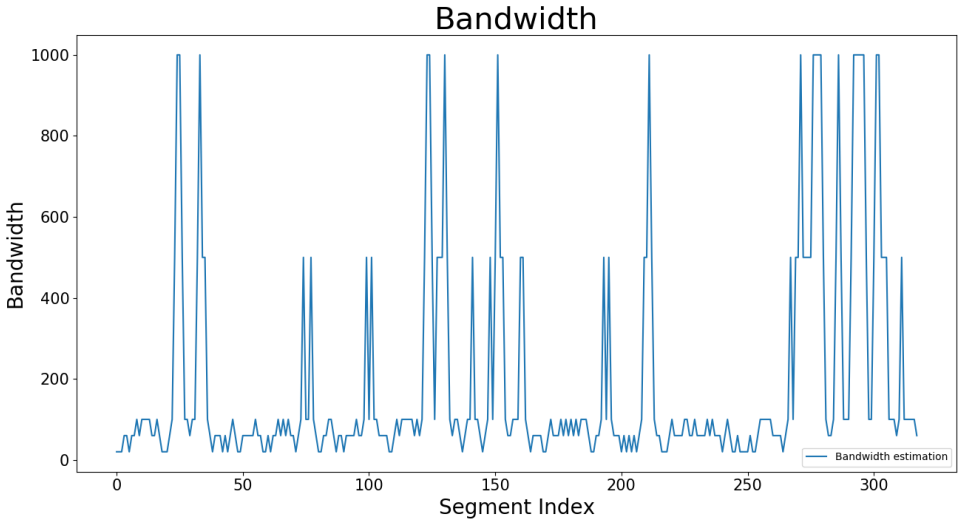


Figure 13: One example of predicted bandwidth

4.1.1. QoE value

For each video, we test 10 different bandwidths with the same bandwidth level. Since the weight $w2$ is base on subjective performance, the value could be change to get proper QoE value. According to equation (1), each QoE value is shown from Table 5 to Table 7.

Table 5: The QoE value for different bandwidth of 'big buck bunny.mp4'

Bandwidth	1	2	3	4	5
QoE	0.5969	1.0968	1.0099	1.1081	1.6105
Bandwidth	6	7	8	9	10
QoE	2.1493	1.1712	1.8167	1.3441	0.9788

For 'Big buck bunny.mp4', $w1 = 0.5,\ w2 = 2$. The average QoE value is 1.2882. One example final request bitrate is shown in Figure 14.

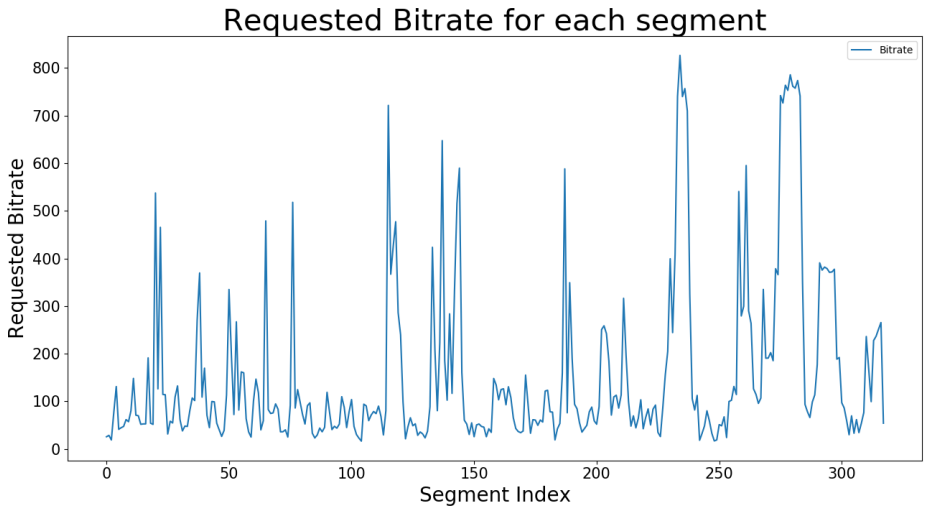


Figure 14: One example of final requested bitrates for $w1 = 0.5,\ w2 = 2$

Table 6: The QoE value for different bandwidth of ‘bear.mp4’

Bandwidth	1	2	3	4	5
QoE	1.8597	2. 1881	2. 329	1. 9191	1. 8331
Bandwidth	6	7	8	9	10
QoE	1. 893	1. 9582	2. 3216	1. 0233	1. 8135

For ‘Bear.mp4’, $w1 = 0.5$, $w2 = 10$. The average QoE value is 1.9139. One example final request bitrate is shown in Figure 15.

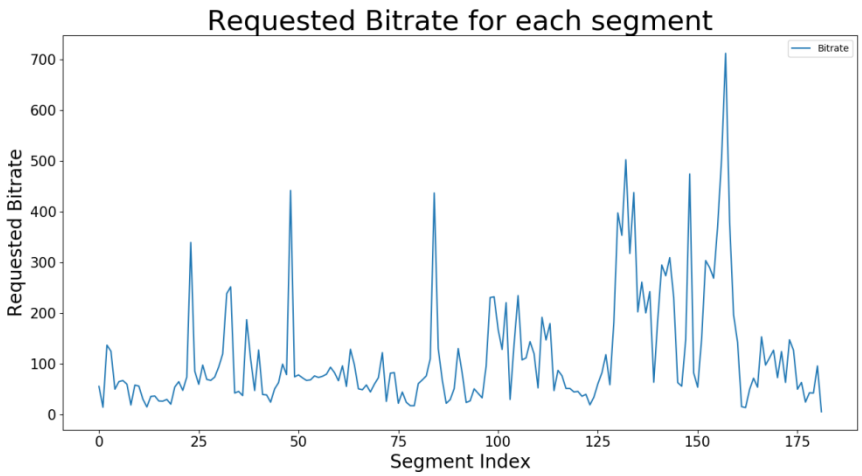


Figure 15: One example of final requested bitrates for $w1 = 0.5$, $w2 = 10$

Table 7: The QoE value for different bandwidth of ‘test.mp4’

Bandwidth	1	2	3	4	5
QoE	2.3604	2.344	1.3422	1.3303	0.9889
Bandwidth	6	7	8	9	10
QoE	1.5684	2.1881	1.6485	1.5866	1.6149

For ‘Test.mp4’, $w1 = 0.5$, $w2 = 3$. The average QoE value is 1.6972. One example final request bitrate is shown in Figure 16.

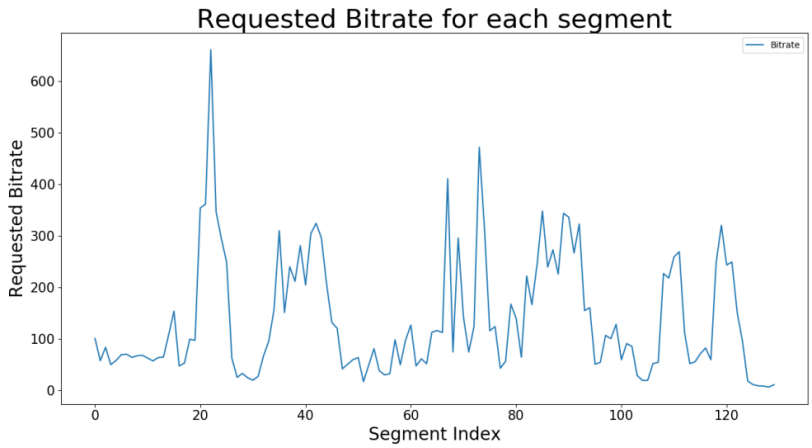


Figure 16: One example of final requested bitrates for $w1 = 0.5$, $w2 = 3$

4.1.2. Different weights

In this part, we take ‘Big buck bunny.mp4’ as the example and change some different weights. Each of them has been tried 10 times. The results are shown in Table 8.

Table 8: The QoE value for different weights

w1	w2	QoE
0.5000	2	1.2882
0.5000	5	0.9261
0.3333	2	1.7039
0.3333	5	1.0172

From Table 8, we can find that $w1$ has less influence on the QoE value, since if $w1$ increases, the QoE value does not decrease as much as when $w2$ increases, which means the quality switching will affect consumers' feeling not very much. It is easy to understand because quality switching could not be keenly aware of by the client and buffer starvation will affect consumers' experience very much.

4.2. Future information

We first consider future $l = 2$ segments. As discussed before, we will introduce internal QoE to decide which quality to choose and we will calculate the final QoE value to compare. Python code is shown in Appendix 6 'base.py'

4.2.1. Bandwidth for future segments stable

For internal QoE, we need to set several weights. First, we set $w1 = \frac{1}{3}$, $w2 = 5,\ \lambda = 0.9,\ l = 2$. The bandwidth is use Appendix 3 'bandwidth.py' to simulate. QoE value is shown as Table 9.

Table 9: The QoE value for stable bandwidth

Bandwidth	1	2	3	4	5
QoE	1.7626	1.9016	1.8322	1.2771	1.5713
Bandwidth	6	7	8	9	10
QoE	1.7044	1.8359	1.6029	1.9226	1.6953

For ‘Big buck bunny.mp4’, $w1 = 0.3333$, $w2 = 2$. The average QoE value is 1.7106.

4.2.2. Bandwidth for future segments variable

Same as before, we set $w1 = \frac{1}{3}$, $w2 = 5$, $\lambda = 0.9$, $l = 2$. The bandwidth is use Appendix 4 ‘bw.py’ to simulate. Each segment has different bandwidth level as Markov chain model. QoE value is shown as Table 10.

Table 10: The QoE value for variable bandwidth

Bandwidth	1	2	3	4	5
QoE	1.8803	1.8204	1.8678	1.8069	1.8076
Bandwidth	6	7	8	9	10
QoE	1.5881	1.7469	1.747	1.7096	1.8954

For ‘Big buck bunny.mp4’, $w1 = 0.3333$, $w2 = 2$. The average QoE value is 1.7870. One example result is shown in Figure 17.

```
(py2) F:\bigbuckbunny>python base.py
Average Quality 3.779874213836478
Quality Variation 0.1924290220820189 2
Starvation Ratio 0.8882072467682434
QoE value 1.939316712939318
```

Figure 17: The screenshot for one example QoE value

4.2.3. Different future segments

We also take video ‘bigbuckbunny.mp4’ as the example. The total number of segments of the video is 318, so we should find the number $l+1$ which can go into 318 without a remainder. In our work, we choose $l=1,\ l=2\ and\ l=5$. Each of them will try 10 different bandwidths. The QoE value is shown in Table 11 and Table 12.

Table 11: The QoE value for $l=1$ future segments

Bandwidth	1	2	3	4	5
QoE	2.0175	2.0044	1.9742	1.9214	2.0964
Bandwidth	6	7	8	9	10
QoE	1.9525	1.9035	2.0270	1.9849	1.9443

For ‘Big buck bunny.mp4’, $w1=0.3333,\ w2=2,\ l=1\ and\ \lambda=0.9$. The average QoE value is 1.9826.

Table 12: The QoE value for $l=5$ future segments

Bandwidth	1	2	3	4	5
QoE	1.1372	1.6458	1.0050	1.9656	1.3450
Bandwidth	6	7	8	9	10
QoE	1.5395	1.5675	1.7181	1.3968	1.4971

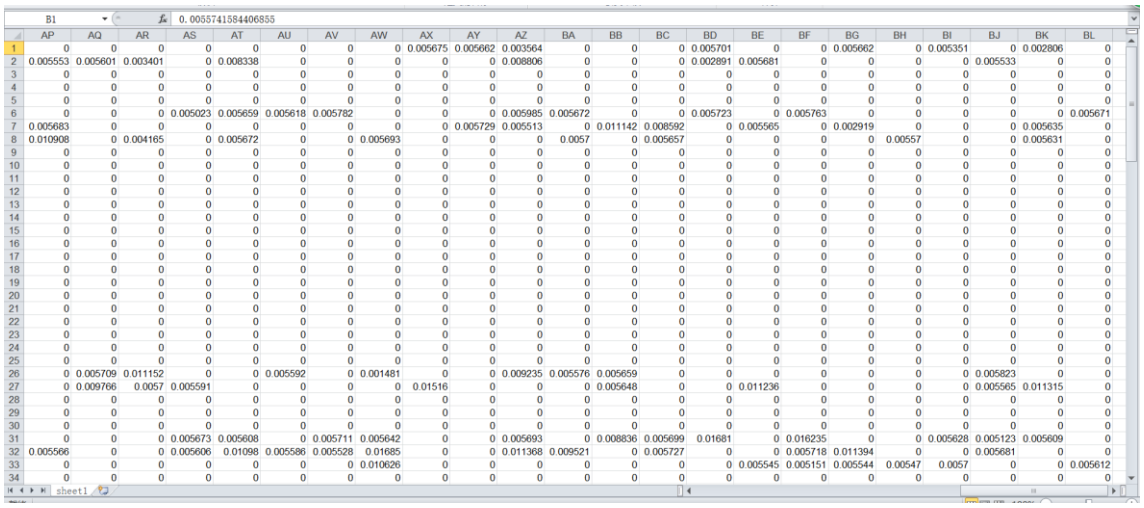
For ‘Big buck bunny.mp4’, $w1=0.3333,\ w2=2,\ l=5\ and\ \lambda=0.9$. The average QoE value is 1.4818.

4.3. Q-learning results

4.3.1. General Q-learning

In order to avoid calculating the instant QoE value each decision point, we train a Q-table to record the proper action according to current states as introduced in Methodology. To compare with the traditional arithmetic method, we also set the $w1 = 0.3333$, $w2 = 2$ and $l = 2$. According to Bellman equation as (26), $\alpha = 0.01$, $\gamma = 0.99$. The part of Q-table is shown in Figure 18.

$$Q(s,a) = Q(s,a) + \alpha * [r + \gamma * max_{\tilde{a}} Q(\tilde{s},\tilde{a}) - Q(s,a)] \qquad (26)$$



	AP	AQ	AR	AS	AT	AU	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL
1	0	0	0	0	0	0	0	0	0.005675	0.005662	0.003564	0	0	0	0.005701	0	0	0.005662	0	0.005351	0	0.002806	0
2	0.005553	0.005601	0.003401	0	0.008338	0	0	0	0	0	0.008806	0	0	0	0.002891	0.005681	0	0	0	0	0.005533	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0.005023	0.005659	0.005618	0.005782	0	0	0	0.005985	0.005672	0	0	0.005723	0	0.005763	0	0	0	0	0	0.005671
7	0.005683	0	0	0	0	0	0	0	0	0.005729	0.005513	0	0.011142	0.008592	0	0.005565	0	0.002919	0	0	0	0.005635	0
8	0.010908	0	0.004165	0	0.005672	0	0	0.005693	0	0	0	0.0057	0	0.005657	0	0	0	0	0.00557	0	0	0.005631	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0.005709	0.011152	0	0	0.005592	0	0.001481	0	0	0.009235	0.005576	0.005659	0	0	0	0	0	0	0	0.005823	0	0
27	0	0.009766	0.0057	0.005591	0	0	0	0	0.01516	0	0	0	0.005648	0	0	0.011236	0	0	0	0	0.005565	0.011315	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0.005673	0.005608	0	0.005711	0.005642	0	0	0.005693	0	0.008836	0.005699	0.01681	0	0.016235	0	0	0.005628	0.005123	0.005609	0
32	0.005566	0	0	0.005606	0.01098	0.005586	0.005528	0.01685	0	0	0.011368	0.009521	0	0.005727	0	0	0.005718	0.011394	0	0	0.005681	0	0
33	0	0	0	0	0	0	0	0.010626	0	0	0	0	0	0	0	0.005545	0.005151	0.005544	0.00547	0.0057	0	0	0.005612
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18: Part of the example Q-table

Different from the traditional method, we trained a Q-table and use the Q-table to test the QoE performance under different bandwidth.

Table 13: The QoE value for algorithm with Q-learning

Bandwdith	1	2	3	4	5
QoE	1.0736	1.0390	1.1867	1.3628	1.0997
Bandwdith	6	7	8	9	10
QoE	1.3152	1.2975	1.1442	1.3409	1.0102

The average QoE value is 1.1870. The requested bitrate is shown in Figure 19.

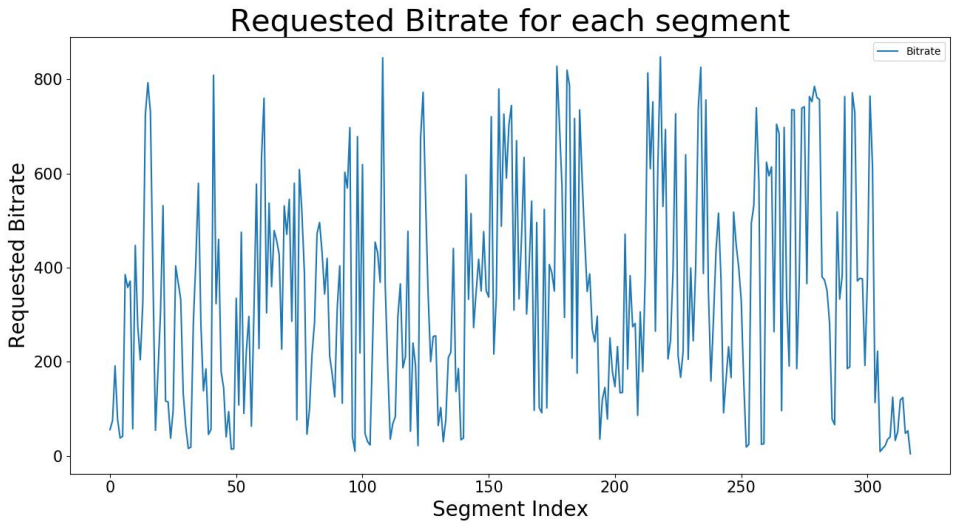


Figure 19: One example of final requested bitrates with Q-learning

4.3.2. Complex Q-learning

As introduced in Methodology, the reward for each state is also based on the bitrates. We build several $Q-table$ to record the proper action according to current states and the current bitrates. To compare with the traditional arithmetic method, we also set the $w1 = 0.3333,\ w2 = 2,\ l =$

2, $\alpha = 0.01$, $\gamma = 0.99$. The part of an example $Q - table$ is shown in Figure 20.

```
[[[0.          0.          0.          ...  0.          0.          0.02055813]
  [0.          0.          0.          ...  0.01077854  0.03131212  0.        ]
  [0.          0.          0.          ...  0.          0.          0.        ]
  ...
  [0.          0.          0.          ...  0.          0.          0.        ]
  [0.          0.          0.          ...  0.          0.035       0.        ]
  [0.          0.          0.02615614  ...  0.          0.035       0.        ]]
```

Figure 20: The screenshot of one of the Q-table

Table 14: The QoE value for algorithm with complex Q-learning

Bandwdith	1	2	3	4	5
QoE	1.2771	1.3234	1.4346	1.3203	1.2671
Bandwdith	6	7	8	9	10
QoE	1.3425	1.2945	1.3110	1.3625	1.3709

The average QoE value is 1.3304. The requested bitrate is shown in Figure 21.

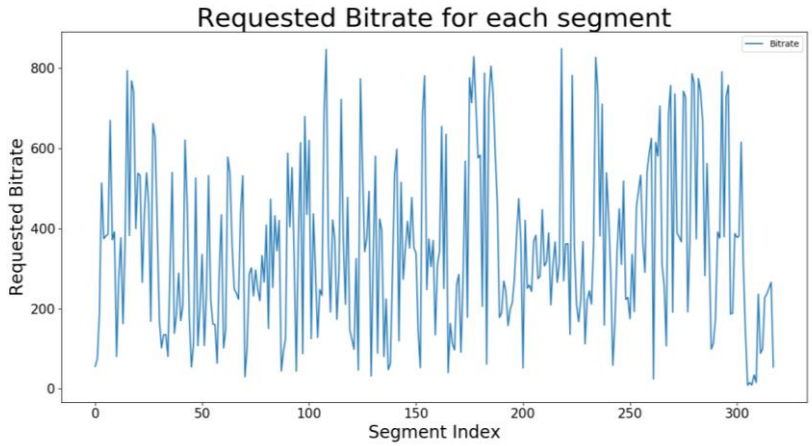


Figure 21: One example of final requested bitrates with complex Q-learning

4.4. Deep Q-learning results

For Deep Q-learning, we use ‘big buck bunny.mp4’and ‘bear.mp4’ to train the network. For the testing, we use the original video and another video to test the network. We tried four different Other weights are the same as before: $w1 = 0.3333,\ w2 = 2,\ \alpha = 0.01,\ \gamma = 0.99.$

When use ‘big buck bunny.mp4’ to train, we set $l = 52$ and the episode number is set 500. If we use itself as the testing video, we set $l = 2$ in order to compare with QoE value before. If we use ‘test.mp4’ as the testing video’, we set $l = 4$. If we use ‘bear.mp4’ as the testing video, we set $l = 6$. The results are shown from Table 15 to Table 17.

Table 15: The QoE value for training video ‘big buck bunny.mp4’ and testing video ‘big buck bunny.mp4’

Bandwdith	1	2	3	4	5
QoE	2.0094	2.3915	1.9979	1.3490	1.3329
Bandwdith	6	7	8	9	10
QoE	1.2177	2.3753	0.7270	2.5510	1.6352

The average QoE value is 1.7587. Figure 22 shows one example for the QoE value. Figure 23 shows the requested bitrates and predicted bandwidth. Figure 24 shows the predicted bandwidth and buffer reservation.

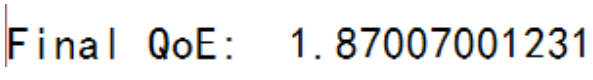


Figure 22: The screenshot of final QoE value

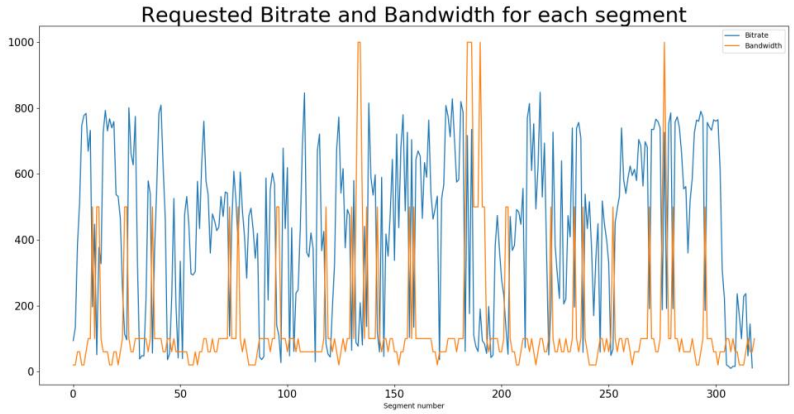


Figure 23: Requested bitrate and estimated bandwidth

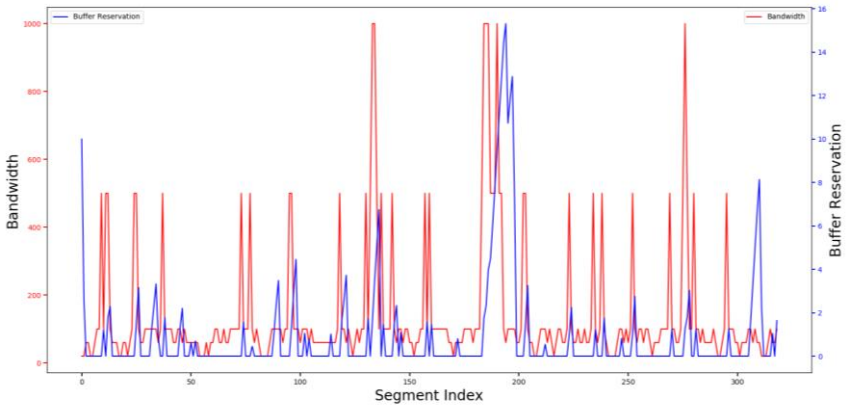


Figure 24: Estimated bandwidth and buffer reservation

Table 16: The QoE value for training video ‘big buck bunny.mp4’ and testing video ‘test.mp4’

Bandwdith	1	2	3	4	5
QoE	1.7689	3.0000	2.1741	2.0000	1.8907
Bandwdith	6	7	8	9	10
QoE	2.7674	4.0000	2.0256	2.0636	0.8405

The average QoE value is 2.2531.

Table 17: The QoE value for training video ‘big buck bunny.mp4’ and testing video ‘bear.mp4’

Bandwdith	1	2	3	4	5
QoE	2.4536	0.7809	2.0000	2.6759	1.0745
Bandwdith	6	7	8	9	10
QoE	1.9569	2.0000	3.7843	3.5335	2.1599

The average QoE value is 2.2415. Then we try to use ‘bear.mp4’ as the training video, we set $l = 25$. For testing, we also use ‘big buck bunny.mp4’ as the testing video, $l = 2$. Use ‘bear.mp4’ as the testing video, $l = 6$. Use ‘test.mp4’ as the testing video, $l = 4$.

Table 18: The QoE value for training video ‘bear.mp4’ and testing video ‘big buck bunny.mp4’

Bandwdith	1	2	3	4	5
QoE	0.0102	0.0301	2.4128	2.9964	1.6577
Bandwdith	6	7	8	9	10
QoE	3.0000	0.9361	0.9996	1.7867	2.5789

The average QoE value is 1.6409.

Table 19: The QoE value for training video ‘bear.mp4’ and testing video ‘bear.mp4’

Bandwdith	1	2	3	4	5
QoE	1.8061	1.9483	2.6216	1.7042	2.7154
Bandwdith	6	7	8	9	10
QoE	2.9905	1.7596	1.8187	0.9519	2.7691

The average QoE value is 2.1085.

Table 20: The QoE value for training video ‘test.mp4’ and testing video ‘test.mp4’

Bandwdith	1	2	3	4	5
QoE	3.0000	2.7675	2.8584	0.8574	1.0392
Bandwdith	6	7	8	9	10
QoE	4.0000	1.4154	2.9782	1.4741	2.3596

The average QoE value is 2.2750

5. Discussion

5.1. Traditional Method

5.1.1. Baseline

Traditional method need to predict the bandwidth and try every quality level to estimate the QoE value in order to find proper playing quality level to get higher QoE value in the client side. As shown before, the average QoE value for baseline is 1.7039. For analysis, request bitrate and estimated bandwidth are shown in Figure 25. Most request bitrate is lower than estimated bandwidth, which will lead to a stable watching quality.

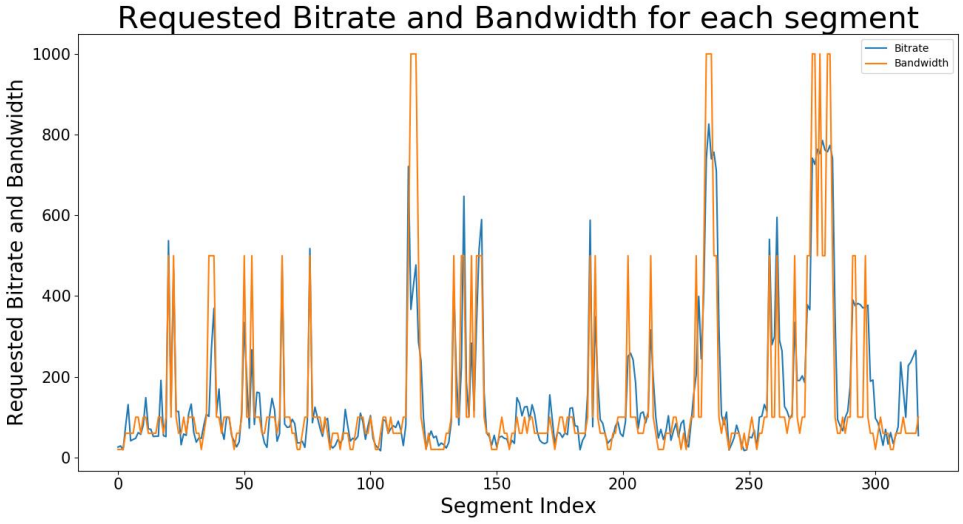


Figure 25: Requested bitrate and estimated bandwidth

Figure 26 shows estimated bandwidth and buffer reservation. From Figure 25 and Figure 26, we can find that if the bandwidth is lower than request bitrate, the buffer reservation will increase. It is easy to understand that if

bandwidth is low, we need more buffer to guarantee a good stable watching quality. If bandwidth is become higher, the buffer reservation becomes decreasing because segments could be downloaded quickly and will take less buffer time.

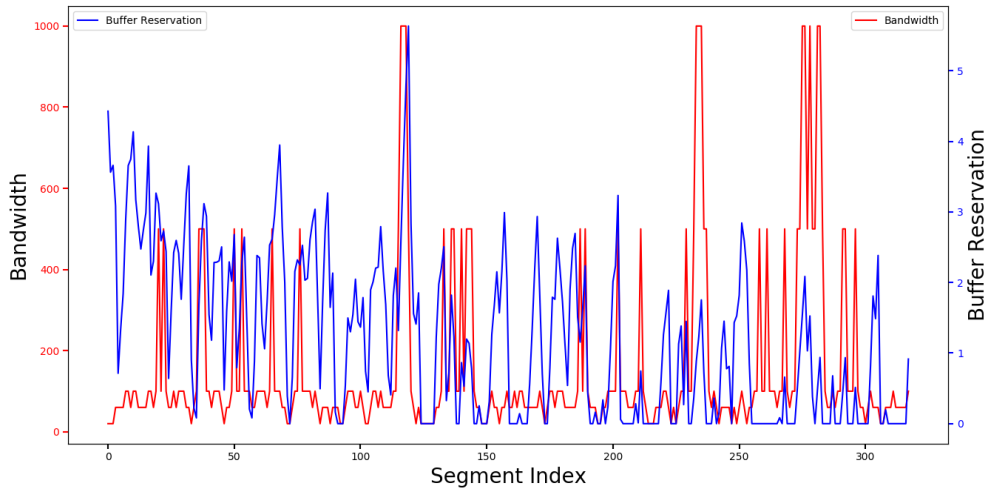


Figure 26: Estimated bandwidth and buffer reservation

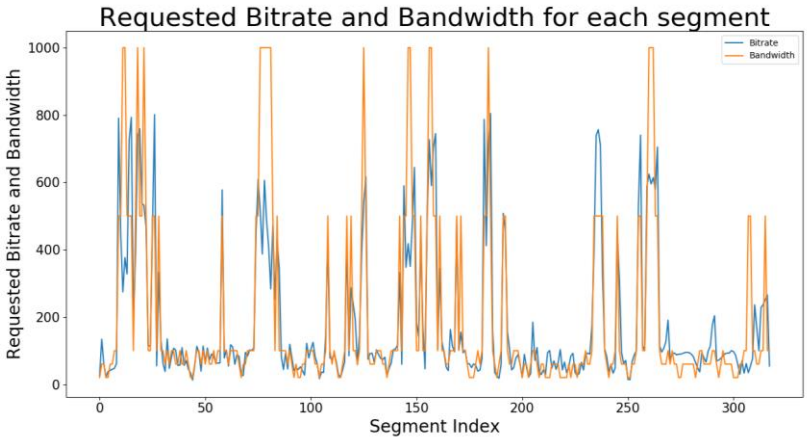
5.1.2. With and without future information

The average QoE value for algorithm with and without future information is shown in Table 21. We can find that average QoE value for algorithm with future information is slightly higher than that of algorithm without future information. It is rational because more information will push the client to make better decisions. In addition, when we introduce the future information, the algorithm with internal QoE will insure that we have less quality variation and find the most possible bandwidth prediction. It is important to consider future information to decide the proper quality level to download.

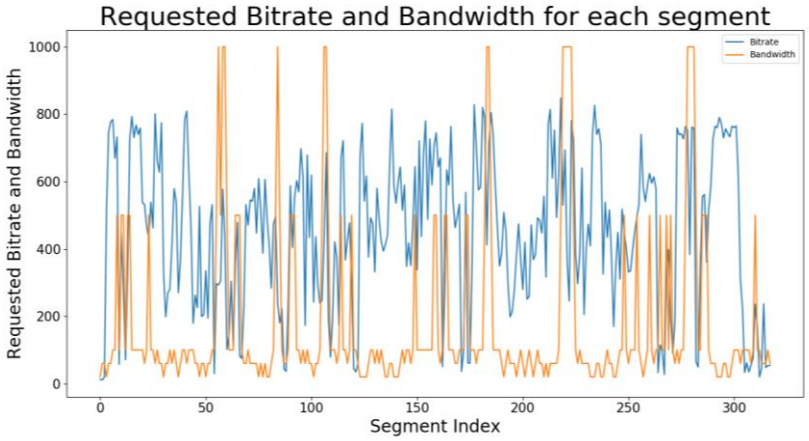
Table 21: The average QoE value for algorithm without and with future information

Without Future Informaion	With Future Information
1.7039	1.7870

The final requested bitrate and the predicted bandwidth for each segment of each algorithm are shown in Figure 27 (a) and (b). We can find that the algorithm without future information make most requested bitrate become lower than the predicted bandwidth, and the bandwidth of the algorithm with algorithm with future information is not always higher than the bitrate. The reason is the buffer reservation will promote the choice of higher quality level. In the same condition, algorithm with future information could get higher watching experience since the high bitrate shown in Figure 27 (b) and without decreasing the QoE value.



(a) Algorithm without future information



(b) Algorithm with future information

Figure 27: The final requested bitrate and the predicted bandwidth for each segment of each algorithm

5.1.3. Stable Bandwidth and Variable Bandwidth

We assume the bandwidth of the future l segments is the same as the bandwidth of current segment, which means the bandwidth will not fluctuate very often, shown in Figure 11 (a). The average QoE value is shown in Table 22.

Table 22: The average QoE value for stable and variable bandwidth

Bandwidth Stable	Bandwidth Variable
1.7106	1.7870

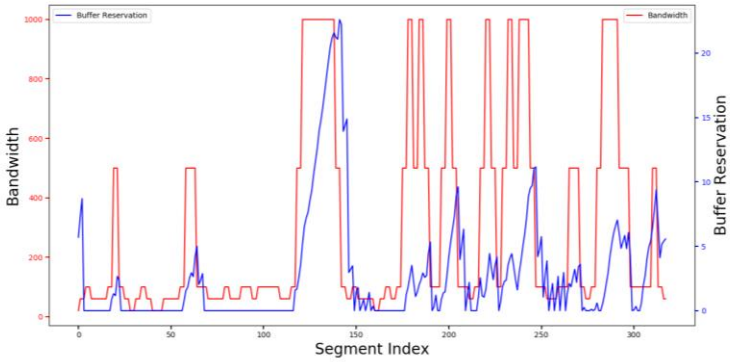
The average QoE value of variable bandwidth is slightly higher than that of stable bandwidth. One example QoE value and the scores of three factors are shown in Table 23. According to Table 23, it is because the situation with stable bandwidth has more quality variation. It is easy to understand

that the bitrates are fluctuating for every segment; if the bandwidth is stable, the quality level will become higher along with the buffer reservation become larger when the size of segment is smaller than the client could download.

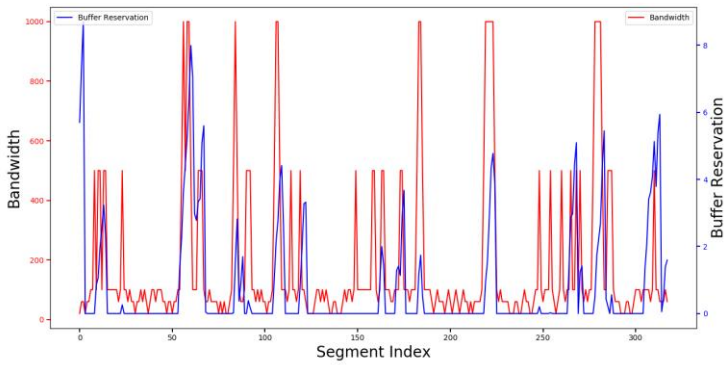
Table 23: QoE performance and score of three factors with different bandwidth

Bandwidth	QoE	Average Quality	Quality Variation	Starvation Ratio
Stable	1.6438	3.4057	0.6215	0.7773
Variable	1.8495	3.6918	0.2744	0.8754

Figure 28 (a) and (b) show the bandwidth and buffer reservation changing with segment. From Figure 28 (a), we can find that the buffer reservation is become larger when the bandwidth is stable at a certain value. If the bandwidth is fluctuating, the buffer reservation will decrease even to 0. This is also shown in Figure 28 (b), where the bandwidth is variable for each segment and the buffer reservation is smaller than Figure 28 (a), so the quality level could not keep changing.



(a) With stable bandwidth



(b) With variable bandwidth

Figure 28: The bandwidth and buffer reservation with different bandwidth

5.1.4. Different Future segments number

The average QoE values for different future segment number are shown in Table 24. We can find that as the number of future segments become larger, the QoE value become smaller. This illustrates that more future information will not bring more contributions because more prediction and estimation may bring more uncertainties so that final performance is not good.

Table 24: The average QoE value for different future segments number

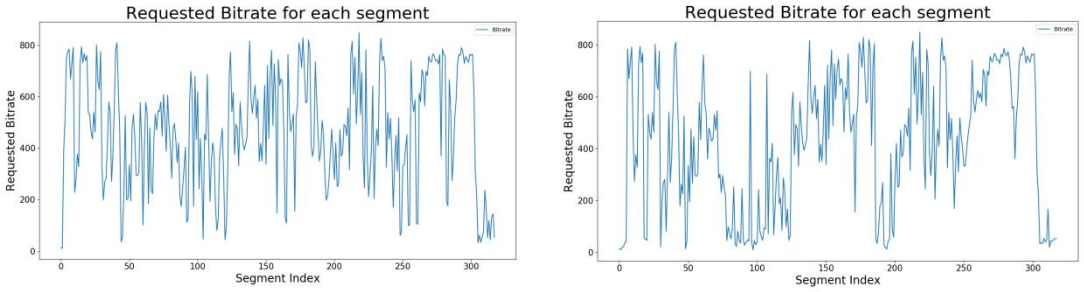
Future segments number	QoE value
l=1	1.9826
l=2	1.7870
l=5	1.4818

The example QoE value and the scores of three factors are shown in Table 25. It demonstrates that the starvation ratio has less contribution to the QoE performance because the value is similar to each other. However, as the

future segment number becomes higher, the average quality becomes lower and the quality variation becomes larger so that the QoE value becomes lower. This means increasing the number of future segment number will increase the quality variation since the uncertainty will increase also. This can also be shown in Figure 29 (a) and (b). The fluctuation degree of Figure 29 (b) is significantly higher than that of Figure 29 (a)

Table 25: QoE performance and score of three factors with different future segments number

Future Segment Number	QoE	Average Quality	Quality Variation	Starvation Ratio
l=1	1.9538	3.7892	0.2303	0.8794
l=2	1.8495	3.6918	0.2744	0.8754
l=5	1.6119	3.4654	0.3060	0.8758



(a) Future segments $l = 1$ (b) Future segments $l = 5$

Figure 29: Requested bitrates for different future information

5.2. Q-learning Method

Traditional method needs to predict each bandwidth and try every possible quality level, it will take too much time and the flexibility of quality level selection strategy is low. It also needs to calculate the instant QoE value to

make decision. In the network environment where bandwidth is variable, quality level switching stability needs to be improved. Machine learning method can enhance the flexibility and adaptability of the client rate decision mechanism. Applying reinforcement learning to the quality selection enables the client to dynamically learn the best behavior in the current network environment.

The average QoE value for traditional method and general Q-learning method is shown in Table 26. For comparison, all the weights are set the same. The QoE value of general Q-learning is lower than that of traditional method because the uncertainty is even higher than in more future information. As shown in Table 27, the quality variation of Q-learning method is higher. It demonstrates that general Q-learning is not suitable for the choice of quality level because as discussed before, the reward function is not only base on the current state (current bandwidth and future predicted bandwidth) and the action (quality level chosen for each segment), but also based on the variable corresponding bitrates for each segment.

Table 26: The average QoE value for different algorithm

Traditional Method	General Q-learning
1.7870	1.1870

Table 27: QoE performance and score of three factors with different algorithm

Method	QoE	Average Quality	Quality Variation	Starvation Ratio
Traditonal Method	1.8495	3.6918	0.2744	0.8754
General Q-learning	1.0584	3.0031	0.8706	0.8272

Based on the shortcomings above, we introduce a complex Q-learning, which is based on different bitrates. Table 28 shows the average QoE value of different method. For comparison, all the weights are set the same. The QoE performance of complex Q-learning is slightly better than the general Q-learning, but still worse than the traditional method because the higher quality variation.

Table 28: The average QoE value for different algorithm

General Q-learning	Complex Q-learning
1.1870	1.3304

Table 29: QoE performance and score of three factors with different algorithm

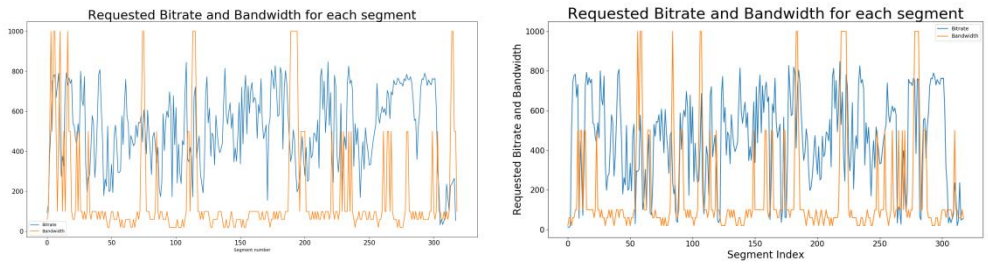
Method	QoE	Average Quality	Quality Variation	Starvation Ratio
Traditonal Method	1.8495	3.6918	0.2744	0.8754
General Q-learning	1.0584	3.0031	0.8706	0.8272
Complex Q-learning	1.3625	3.3050	0.6845	0.8571

Since both method based on Q-learning is not as good as the traditional method, we finally introduce Deep Q-learning method to experiment. For comparison, all the weights are set the same. Because the Deep Q-learning includes two parts: training and testing, to get the corresponding QoE value, we first also use the trained network to test the ‘big buck bunny.mp4’. The results are shown in Table 30.

Table 30: The average QoE value for different algorithm

Traditional Method	General Q-learning	Complex Q-learning	Deep Q-learning
1.7870	1.1870	1.3304	1.7587

From Table 30, the QoE performance which used Deep Q-learning is better than the results that use simple Q-learning and is similar to the result that use traditional method. Figure 30 shows the requested bitrate and predicted bandwidth of traditional method and Deep Q-learning algorithm. We can find that the requested the degree of variation of requested bitrate is similar. In addition, Deep Q-learning provides a trained network so that we do not need to try every possible quality level according to the predicted bandwidth as traditional method process. For the client, we just need to input the bitrates of different quality level and the possible bandwidth; after that we can get the proper playing quality level so that the client will have excellent playing experience.



(a) With traditional method (b) With Deep Q-learning

Figure 30: Requested bitrate and predicted bandwidth of different method

We also try to use the trained network to test other videos; the results are shown in Table 31.

Table 31: The average QoE value with training video ‘big buck bunny.mp4’ for different testing video

Big buck bunny	Test	Bear
1.7587	2.2531	2.2415

The trained network also gets good QoE performance on other video. The reason that the average QoE value for ‘test.mp4’ and ’bear.mp4’ is higher maybe the bitrates of each video are similar and the length of ‘big buck bunny,mp4’ is longer than other videos. Use longer videos to train the network lead to more proper network. The average QoE results are shown in Table 32.

Table 32: The average QoE value with different training videos for different testing video

Training video	Testing video	QoE
Big buck bunny	Big buck bunny	1.7587
	Bear	2.2415
	Test	2.2531
Bear	Big buck bunny	1.6409
	Bear	2.1085
	Test	2.2750

If we use longer video as the training network, the QoE performance is better, which is shown in Line 2, 3 and 6. But if we use shorter video to train the network, the QoE performance is not good, which is shown in

Line 4, the length of ‘bear.mp4’ is shorter than that of ‘big buck bunny.mp4’.

6. Conclusion

In this paper, we quantified the client viewing experience into a concrete calculation and discussed three factors that affect QoE, which means a suitable selection of different video quality of proper representation to download. From our experiments, we can find that different factors have different contributions to consumers' experience. We discussed that the weights of the three factors have different degrees of influence on QoE performance. The loss caused by quality switching during video playing has little influence on watching experience. Buffer reservation and starvation has a great impact on QoE. In addition, future information will affect experience.

After that, we fixed the weights of the three factors and try to find proper method to get better QoE performance. First, we introduce future information to make wiser decisions for quality level. It is important to using future information but more future information could not make wiser decision. We also experiment on different type of bandwidth to judge the quality level it chose. It considered the bandwidth in the client and the buffer reservation. Stable bandwidth and long buffer reservation will lead to more quality switching loss.

Traditional method needs to calculate the instant QoE value to make decision for quality level choosing. It takes time and less flexibility. After that, we introduce other method to make the decision of quality level for each segment. To compare, we also calculate the final QoE value with the decided quality level for each segment. When use simple Q-learning, the

QoE performance is not as good as the traditional method although we provide a complex Q-learning method. Simple Q-learning is not suitable for quality level decision.

In the end, we introduce the Deep Q-learning method and build a simple network. We need to input the bitrates of different quality level and the possible bandwidth; after that we can get the proper playing quality level so that the client will have excellent playing experience. We also try to use the trained network to test other videos. The QoE performance is no less than traditional method. The training video should be longer than the testing video in order to make wiser decisions on quality choosing.

7. Future work

Now that we understand the importance of the weight of the three factors affecting QoE and the importance of introducing future information, to make wiser decision on quality level choosing based on the QoE performance, better neural network need to be built and modified. Also, the improvements of DQN methods could be introduced. However, DQN cannot output the probability of actions, so the Policy Network is a better method. DQN is a value-based method. In other words, it selects the most valuable action to execute by calculating the value of each state action. It's more general to use the output probability. A more direct approach is to update the Policy Network. A Policy Network is a neural Network in which the input is the state and the output is the action (not the Q value).

Reference

[1] M. Claeys, S. Latré, J. Famaey, T. Wu, W. V. Leekwijck, and F. D. Turck, "Design of a Q-Learning based client quality selection algorithm for HTTP Adaptive Video Streaming," in *Adaptive and Learning Agents Workshop*, 2013, pp. 30-37.

[2] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469-492, 2017.

[3] T. C. Thang, Q. D. Ho, J. W. Kang, and A. T. Pham, "Adaptive streaming of audiovisual content using MPEG DASH," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 78-85, 2012.

[4] J. Kua, G. Armitage, and P. Branch, "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1842-1866, 2017.

[5] T. Stockhammer, "Dynamic adaptive streaming over HTTP --:standards and design principles," in *ACM Conference on Multimedia Systems*, 2011, pp. 133-144.

[6] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *J Artificial Intelligence Research*, vol. 4, no. 1, pp. 237--285, 1996.

[7] L. R. Xiong, J. Z. Lei, and X. Jin, "Research on Q-learning based rate control approach for HTTP adaptive streaming," *Journal on Communications*, 2017.

[8] Young_Gy. (2017). Available: https://blog.csdn.net/Young_Gy/article/details/73485518#q-learning

[9] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," *Machine Learning*, vol. 8, no. 3-4, pp. 225-227, 1992.

[10] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *Computer Science*, 2013.

[11] T. C. Thang, H. T. Le, A. T. Pham, and M. R. Yong, "An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 693-705, 2014.

[12] L. W. Chen, L. I. Guo-Ping, G. W. Teng, H. W. Zhao, and G. Z. Wang, "Adaptive Hybrid Rate Control Algorithm Based on HTTP Streaming," *Journal of Shanghai University*, 2014.

[13] C. Mueller, S. Lederer, and C. Timmerer, "An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments," in *ACM Sigmm Workshop on Mobile Video*, 2012, pp. 37-42.

[14] H. T. Le, D. V. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Buffer-based bitrate adaptation for adaptive HTTP streaming," in *International Conference on Advanced Technologies for Communications*, 2014, pp. 33-38.

[15] L. R. Xiong, J. Z. Lei, and X. Jin, "Hybrid Rate Adaptation Algorithm for Adaptive HTTP Streaming," *COMPUTER SCIENCE*, vol. 44, no. 2, p. 5, 2017.

[16] Z. Li *et al.*, "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719-733, 2014.

[17] S. Hesse, "Design of scheduling and rate-adaptation algorithms for adaptive HTTP streaming," in *SPIE Optical Engineering + Applications*, 2013, p. 88560M.

[18] W. Wu, M. A. Arefin, R. Rivas, K. Nahrstedt, R. M. Sheppard, and Z. Yang, "Quality of experience in distributed interactive multimedia environments: Toward a theoretical framework," in *ACM International Conference on Multimedia*, 2009, pp. 481-490.

[19] R. K. P. Mok, E. W. W. Chan, and R. K. C. Chang, "Measuring the Quality of Experience of HTTP Video Streaming," in *Ifip/ieee International Symposium on Integrated Network Management*, 2011, pp. 485-492.

[20] T. Hoßfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia, "Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies," *Computer Networks*, vol. 81, no. C, pp. 320-332, 2015.

[21] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "QDASH:a QoE-aware DASH system," in *Multimedia Systems Conference*, 2012.

[22] D. Silver. (2016). *Tutorial: Deep Reinforcement Learning*. Available: https://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

[23] V. Martín, J. Cabrera, and N. García, "Evaluation of Q-Learning approach for HTTP adaptive streaming," in *IEEE International Conference on Consumer Electronics*, 2016, pp. 293-294.

[24] R. S. H. Istepanian, N. Y. Philip, and M. G. Martini, *Medical QoS provision based on reinforcement learning in ultrasound streaming over 3.5G wireless systems*. IEEE Press, 2009, pp. 566-574.

[25] M. Claeys, S. Latre, J. Famaey, and F. D. Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming

Client," *IEEE Communications Letters,* vol. 18, no. 4, pp. 716-719, 2014.

[26] V. Mart ń, J. Cabrera, and N. Garc á, "Q-learning based control algorithm for HTTP adaptive streaming," in *Visual Communications and Image Processing*, 2016, pp. 1-4.

[27] S. Lederer and C. Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *ACM Sigmm Conference on Multimedia Systems, Mmsys 2012, Chapel Hill, Nc, Usa, February*, 2012, pp. 89-94.

[28] Available: https://tv.sohu.com/v/MjAxMTA5MjgvbjMyMDgyNTUzMy5zaHRtbA==.html

[29] H. Zhang. (2016). Available: https://blog.csdn.net/nonmarking/article/details/50397153

[30] L. Yu, T. Tillo, and J. Xiao, "QoE-Driven Dynamic Adaptive Video Streaming Strategy With Future Information," *IEEE Transactions on Broadcasting*, vol. PP, no. 99, pp. 1-12, 2017.

[31] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *Computer Science*, 2014.

Appendix

Appendix 1: ‘fps.py’

```
import os
import cv2
from sys import argv
script, filename = argv

if __name__ == '__main__' :

    video = cv2.VideoCapture(filename);

    # Find OpenCV version
    (major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')

    if int(major_ver)  < 3 :
        fps = video.get(cv2.cv.CV_CAP_PROP_FPS)
        print ("Frames per second using video.get(cv2.cv.CV_CAP_PROP_FPS):
        {0}".format(fps))
    else :
        fps = video.get(cv2.CAP_PROP_FPS)
        print "Frames per second using video.get(cv2.CAP_PROP_FPS) : {0}".format(fps)

    video.release()

```

Appendix 2: ‘bitrate.py’

```
#C:\DASHEncoder2-dev-v2
import os
import os.path as path
import glob
import re
import numpy as np
import xlwt
from sys import argv
script, segmentsize = argv

###  Search the segments path
filepath=path.abspath(path.dirname(__file__))

MEDIA_FILE        = 'dashtest'
VIDEO_DIR         = 'video'

segment_path=path.join(filepath,MEDIA_FILE,VIDEO_DIR)
dirlist=os.listdir(segment_path)



### Find the number of segments and quality level
dirnum=0
segnum=0

for dirname in dirlist:
    if path.isdir(segment_path):
        dirnum += 1
    seglist = os.listdir(path.join(segment_path,dirname))
for segname in seglist:
    ext = path.splitext(segname)[1]
    if ext == '.m4s':
        segnum=segnum+1
print 'Number of Quality level',dirnum
print 'Number of Segment',segnum


### Calculate the segment size
def get_FileSize(filePath):
    #filePath = unicode(filePath,'utf8')
    fsize = os.path.getsize(filePath)
    fsize = fsize/float(1024)
    return round(fsize,2)


### Calculate the bitrate for each segment
bitrate=np.zeros((dirnum,segnum))

for dirname in dirlist:
    seglist = os.listdir(path.join(segment_path,dirname))
    i=int(filter(str.isdigit,dirname))
    #print i
    for segname in seglist:
        ext = path.splitext(segname)[1]
        if ext == '.m4s':
            j=int(filter(str.isdigit,path.splitext(segname)[0]))
            #print j
            bitrate[i-1][j-1] =
            get_FileSize(path.join(segment_path,dirname,segname))/float(segmentsize)



### Save the bitrates as excel file
book = xlwt.Workbook()
sheet = book.add_sheet(u'sheet1',cell_overwrite_ok=True)

for i in xrange(0,segnum):
    for j in xrange(0,dirnum):
        sheet.write(i,j,bitrate[j][i])
        #sheet.write(i,0,i)
        #sheet.write(0,j,j)
```

Appendix 3: ‘bandwidth.py’

```
import numpy as np
import random
import xlwt
import os.path as path
import matplotlib.pyplot as plt
from sys import argv
script, segnum, futuresegment, bandwidthlevel = argv

### Input the bandwidth level
l=int(futuresegment)
bwlevel = bandwidthlevel.split(',')
bwlevel=[int(x) for x in bwlevel]
print bwlevel
levelnum = len(bwlevel)
print levelnum


### The transition matrix

#A=[0.4  0.6   0    0    0
#   0.2  0.4  0.4   0    0
#    0   0.4  0.4  0.2   0
#    0    0   0.5  0.3  0.2
#    0    0    0   0.5  0.5]


### Predict the next segment bandwidth according to Markov chain
bandwidth = np.zeros((1,int(segnum)))
bandwidth[0][0]=bwlevel[0]
for i in xrange(1,int(segnum),l+1):
    if bandwidth[0][i-1] == bwlevel[0]:
        bandwidth[0][i] = int(np.random.choice([bwlevel[0],bwlevel[1]],1,p=[0.4,0.6]))
        bandwidth[0][i:i+l+1] = bandwidth[0][i]
    elif bandwidth[0][i-1] == bwlevel[1]:
        bandwidth[0][i] =
        int(np.random.choice([bwlevel[0],bwlevel[1],bwlevel[2]],1,p=[0.2,0.4,0.4]))
        bandwidth[0][i:i+l+1] = bandwidth[0][i]

    elif bandwidth[0][i-1] == bwlevel[2]:
        bandwidth[0][i] =
        int(np.random.choice([bwlevel[1],bwlevel[2],bwlevel[3]],1,p=[0.4,0.4,0.2]))
        bandwidth[0][i:i+l+1] = bandwidth[0][i]
    elif bandwidth[0][i-1] == bwlevel[3]:
        bandwidth[0][i] =
        int(np.random.choice([bwlevel[2],bwlevel[3],bwlevel[4]],1,p=[0.5,0.3,0.2]))
        bandwidth[0][i:i+l+1] = bandwidth[0][i]
    elif bandwidth[0][i-1] == bwlevel[4]:
        bandwidth[0][i] = int(np.random.choice([bwlevel[3],bwlevel[4]],1,p=[0.5,0.5]))
        bandwidth[0][i:i+l+1] = bandwidth[0][i]
print bandwidth


### Save the bandwidth as excel file
book = xlwt.Workbook()
sheet = book.add_sheet(u'sheet1',cell_overwrite_ok=True)
filepath=path.abspath(path.dirname(__file__))

MEDIA_FILE        = 'dashtest'
VIDEO_DIR         = 'video'

segment_path=path.join(filepath,MEDIA_FILE,VIDEO_DIR)
for i in xrange(0,int(segnum)):
    sheet.write(i,1,bandwidth[0][i])
    sheet.write(i,0,i+1)

book.save(path.join(segment_path,'bandwidth.xls'))

### Plot the estimated bandwidth
plt.figure()
x=np.arange(0,318)
plt.plot(x,bandwidth[0],label='Bandwidth estimation')
plt.xlabel('Segment Index',fontsize=20)

plt.ylabel('Bandwidth',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Bandwidth',fontsize=30)
plt.legend()
plt.show()

```

Appendix 4: ‘bw.py’

```
import numpy as np
import random
import xlwt
import os.path as path
import matplotlib.pyplot as plt
from sys import argv
script, segnum, bandwidthlevel = argv

### Input the bandwidth level
bwlevel = bandwidthlevel.split(',')
bwlevel=[int(x) for x in bwlevel]
print bwlevel
levelnum = len(bwlevel)
print levelnum


### The transition matrix

#A=[0.4  0.6   0    0    0
#   0.2  0.4  0.4   0    0
#    0   0.4  0.4  0.2   0
#    0    0   0.5  0.3  0.2
#    0    0    0   0.5  0.5]


### Predict the next segment bandwidth according to Markov chain
bandwidth = np.zeros((1,int(segnum)))
bandwidth[0][0]=bwlevel[0]
for i in xrange(1,int(segnum)):
    if bandwidth[0][i-1] == bwlevel[0]:
        bandwidth[0][i] = int(np.random.choice([bwlevel[0],bwlevel[1]],1,p=[0.4,0.6]))
        #bandwidth[0][i:i+l+1] = bandwidth[0][i]
    elif bandwidth[0][i-1] == bwlevel[1]:
        bandwidth[0][i] =
        int(np.random.choice([bwlevel[0],bwlevel[1],bwlevel[2]],1,p=[0.2,0.4,0.4]))
        #bandwidth[0][i:i+l+1] = bandwidth[0][i]

    elif bandwidth[0][i-1] == bwlevel[2]:
        bandwidth[0][i] =
        int(np.random.choice([bwlevel[1],bwlevel[2],bwlevel[3]],1,p=[0.4,0.4,0.2]))
        #bandwidth[0][i:i+l+1] = bandwidth[0][i]
    elif bandwidth[0][i-1] == bwlevel[3]:
        bandwidth[0][i] =
        int(np.random.choice([bwlevel[2],bwlevel[3],bwlevel[4]],1,p=[0.5,0.3,0.2]))
        #bandwidth[0][i:i+l+1] = bandwidth[0][i]
    elif bandwidth[0][i-1] == bwlevel[4]:
        bandwidth[0][i] = int(np.random.choice([bwlevel[3],bwlevel[4]],1,p=[0.5,0.5]))
        #bandwidth[0][i:i+l+1] = bandwidth[0][i]
print bandwidth


### Save the bandwidth as excel file
book = xlwt.Workbook()
sheet = book.add_sheet(u'sheet1',cell_overwrite_ok=True)
filepath=path.abspath(path.dirname(__file__))

MEDIA_FILE       = 'dashtest'
VIDEO_DIR        = 'video'

segment_path=path.join(filepath,MEDIA_FILE,VIDEO_DIR)
for i in xrange(0,int(segnum)):
    sheet.write(i,1,bandwidth[0][i])
    sheet.write(i,0,i+1)


book.save(path.join(segment_path,'bw.xls'))


### Plot the estimated bandwidth
plt.figure()
x=np.arange(0,318)
plt.plot(x,bandwidth[0],label='Bandwidth estimation')

plt.xlabel('Segment Index',fontsize=20)
plt.ylabel('Bandwidth',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Bandwidth',fontsize=30)
plt.legend()
plt.show()
```

Appendix 5: ‘test.py’

```
import os
import os.path as path
import glob
import matplotlib.pyplot as plt
import numpy as np
import xlwt
import xlrd



### Search the segments path
filepath=path.abspath(path.dirname(__file__))

MEDIA_FILE        = 'dashtest'
VIDEO_DIR         = 'video'

segment_path=path.join(filepath,MEDIA_FILE,VIDEO_DIR)

### Open the bitrate and bandwdith file
bitratebook=xlrd.open_workbook(path.join(segment_path,'bigbuckbunny.xls'))
bandwidthbook=xlrd.open_workbook(path.join(segment_path,'bw.xls'))

bitratedata=bitratebook.sheets()[0]
bwdata=bandwidthbook.sheets()[0]


### Calculate the number of quality level and segment
qualitynum=bitratedata.ncols
segnum=bitratedata.nrows


### Write the data into lists
bitrate=[]
bw=[]

for rownum in xrange(0,segnum):
    for colnum in xrange(0,qualitynum):
        bitrate.append(bitratedata.cell(rowx=rownum,colx=colnum).value)
bitrate=[bitrate]


br=np.zeros((segnum,qualitynum))
for i in xrange(0,segnum):
    for j in xrange(0,qualitynum):
        br[i][j]=float(bitrate[0][i*qualitynum+j])
#print br

for rownum in xrange(0,segnum):
    bw.append(bwdata.cell(rowx=rownum,colx=1).value)
bw=[bw]
#print bw

### set the duration of each segment
duration=2

### set the weights of QoE
#w1=0.5
w1=0.33333333333333
w2=2.0000000000000
#w2=5.0000000000000000


### Calculate QoE
a=np.zeros((1,segnum))
T=np.zeros((1,segnum+1))
Q=np.zeros((1,segnum))
Ts=np.zeros((1,segnum))
quality=np.zeros((1,segnum))
quality[0][0]=2
QOE=np.zeros((segnum,4))
QoE=np.zeros((1,segnum))
Star=np.zeros((1,segnum))
```

```
T[0][0]=5
for i in xrange(1,segnum):
    Q[0][0]=2
    T[0][1]=max(T[0][0]-br[0][int(Q[0][0])-1]*duration/bw[0][0]+duration,0)
    Ts[0][0]=max(br[0][int(Q[0][0])-1]*duration/bw[0][0]-T[0][0],0)
    a[0][0]=br[0][int(Q[0][0])-1]*duration/bw[0][0]
    if i>0: #i=1
### Try the lowest quality level
        quality[0][i]=1
        S=sum(quality[0])
        E=S/(i+1)
        v=np.zeros((1,i))
        for j in xrange(0,i):
            v[0][j]=abs(quality[0][j+1]-quality[0][j])
        V=sum(v[0])/i
        Ts[0][i]=max(br[i][0]*duration/bw[0][i]-T[0][i],0)
        ts=sum(Ts[0])
        Tt=(i+1)*duration+ts
        Ps=ts/Tt
        QOE[i][0]=E-w1*V-w2*Ps
### Try the second lowest quality level
        quality[0][i]=2
        S=sum(quality[0])
        E=S/(i+1)
        v=np.zeros((1,i))
        for j in xrange(0,i):
            v[0][j]=abs(quality[0][j+1]-quality[0][j])
        V=sum(v[0])/i
        Ts[0][i]=max(br[i][1]*duration/bw[0][i]-T[0][i],0)
        ts=sum(Ts[0])
        Tt=(i+1)*duration+ts
        Ps=ts/Tt
        QOE[i][1]=E-w1*V-w2*Ps
### Try the second highest quality level
        quality[0][i]=3
        S=sum(quality[0])
        E=S/(i+1)
        v=np.zeros((1,i))
        for j in xrange(0,i):
            v[0][j]=abs(quality[0][j+1]-quality[0][j])
        V=sum(v[0])/i
        Ts[0][i]=max(br[i][2]*duration/bw[0][i]-T[0][i],0)
        ts=sum(Ts[0])
        Tt=(i+1)*duration+ts
        Ps=ts/Tt
        QOE[i][2]=E-w1*V-w2*Ps
### Try the highest quality level
        quality[0][i]=4
        S=sum(quality[0])
        E=S/(i+1)
        v=np.zeros((1,i))
        for j in xrange(0,i):
            v[0][j]=abs(quality[0][j+1]-quality[0][j])
        V=sum(v[0])/i
        Ts[0][i]=max(br[i][3]*duration/bw[0][i]-T[0][i],0)
        ts=sum(Ts[0])
        Tt=(i+1)*duration+ts
        Ps=ts/Tt
        QOE[i][3]=E-w1*V-w2*Ps
### Find the quality that makes the QoE best
    qoe=QOE[i,:]
    Q[0][i]=np.argmax(qoe,axis=0)+1
    a[0][i]=br[i][int(Q[0][i])-1]*duration/bw[0][i]
    T[0][i+1]=max(T[0][i]-br[i][int(Q[0][i])-1]*duration/bw[0][i]+duration,0)
    Ts[0][i]=max(br[i][int(Q[0][i])-1]*duration/bw[0][i]-T[0][i],0)
    quality[0][i]=Q[0][i]

print Q

### Calculate the final QoE value
Average=np.mean(Q)
print Average
```

```
varies=np.zeros((1,segnum-1))

for k in xrange(0,segnum-1):
    varies[0][k]=abs(Q[0][k+1]-Q[0][k])
Var=np.sum(varies)/(segnum-1)
print Var

TS=np.sum(Ts)
TT=duration*segnum+TS
PS=TS/TT
QoE=Average-w1*Var-w2*PS
print PS
print QoE


### Plot the figures
fig = plt.figure(1)
x=np.arange(0,segnum)
bit=np.zeros((1,segnum))
for i in xrange(0,segnum):
    bit[0][i]=br[i][int(Q[0][i])-1]
ax=fig.add_subplot(111)
Bitrate,=ax.plot(x,bit[0],label='Bitrate')
Bandwidth,=ax.plot(x,bw[0],label='Bandwidth')
plt.xlabel('Segment Index',fontsize=20)
plt.ylabel('Requested Bitrate and Bandwidth',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Requested Bitrate and Bandwidth for each segment',fontsize=30)
ax.legend()


plt.figure(2)
plt.plot(x,bit[0],label='Bitrate')
plt.xlabel('Segment Index',fontsize=20)
plt.ylabel('Requested Bitrate',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Requested Bitrate for each segment',fontsize=30)
plt.legend()


fig,left_axis=plt.subplots()
#fig.subplots_adjust(right_axis=0,75)
right_axis = left_axis.twinx()

p1, = left_axis.plot(x, bw[0],color='red',label='Bandwidth')
p2, = right_axis.plot(x, T[0][1:],color='blue',label='Buffer Reservation')

left_axis.set_xlabel('Segment Index',fontsize=20)
left_axis.set_ylabel('Bandwidth',fontsize=20)
right_axis.set_ylabel('Buffer Reservation',fontsize=20)

tkw = dict(size=5, width=1.5)
left_axis.tick_params(axis='y', colors=p1.get_color(), **tkw)
right_axis.tick_params(axis='y', colors=p2.get_color(), **tkw)
left_axis.tick_params(axis='x', **tkw)

left_axis.legend()
right_axis.legend(loc='upper left')


plt.show()
```

Appendix 6: ‘base.py’

```
import os
import os.path as path
import glob
import matplotlib.pyplot as plt
import numpy as np
import xlwt
import xlrd
from itertools import *



### Search the segments path
filepath=path.abspath(path.dirname(__file__))

MEDIA_FILE        = 'dashtest'
VIDEO_DIR         = 'video'

segment_path=path.join(filepath,MEDIA_FILE,VIDEO_DIR)

### Open the bitrate and bandwdith file
bitratebook=xlrd.open_workbook(path.join(segment_path,'bigbuckbunny.xls'))
bandwidthbook=xlrd.open_workbook(path.join(segment_path,'bw.xls'))

bitratedata=bitratebook.sheets()[0]
bwdata=bandwidthbook.sheets()[0]

### Calculate the number of quality level and segment
qualitynum=bitratedata.ncols
segnum=bitratedata.nrows

### Write the data into lists
bitrate=[]
bw=[]

for rownum in xrange(0,segnum):
    for colnum in xrange(0,qualitynum):
        bitrate.append(bitratedata.cell(rowx=rownum,colx=colnum).value)
bitrate=[bitrate]

br=np.zeros((segnum,qualitynum))
for i in xrange(0,segnum):
    for j in xrange(0,qualitynum):
        br[i][j]=float(bitrate[0][i*qualitynum+j])
#print br

for rownum in xrange(0,segnum):
    bw.append(bwdata.cell(rowx=rownum,colx=1).value)
bw=[bw]
#print bw

### Future segments number
l=2

### Duration of each segment
duration=2

### Every possible quality level and bandwdith for current and future segment
fi=list(product(xrange(1,5),repeat=l+1))
theta=list(product(xrange(1,6),repeat=l+1))
for i in xrange(0,len(fi)):
    fi[i]=list(fi[i])
for i in xrange(0,len(theta)):
    theta[i]=list(theta[i])
#print len(fi)



### Weights of QoE
w1=0.3333333333333333
w2=2
lamda=0.9

```

```
### Bandwidth level
bwlevel=[20,60,100,500,1000]


### Transition matrix
A=[[0.4,0.6,0,0,0],[0.4,0.4,0.2,0,0],[0,0.2,0.4,0.4,0],[0,0,0.5,0.3,0.2],[0,0,0,0.5,0.5]]




### Find the index in bandwdith level of predicted bandwdith
Bw=np.zeros((1,segnum))
for i in xrange(0,segnum):
    Bw[0][i]=int(bwlevel.index(bw[0][i])+1)

t=np.zeros((segindex+1,l+1))
ts=np.zeros((segindex,l+1))
### Set the prebuffer
t[0][1]=5
qoe=np.zeros((segindex,len(fi)))
segindex=segnum/(l+1)
Q=[]
BW=[]

for i in xrange(0,segindex):
    seg=i*(l+1)
    for j in xrange(0,len(fi)):
        q=fi[j][:]
        e=np.mean(q)
        v=np.zeros((1,len(q)-1))
        for k in xrange(0,len(q)-1):
            v[0][k]=abs(q[k+1]-q[k])
        vv=np.sum(v)/(len(q)-1)
        t[i+1][0]=max(t[i][1]-br[seg][int(q[0])-1]*duration/bw[0][seg]+duration,0)
        for x in xrange(0,l):
            t[i+1][x+1]=max(t[i+1][x]-br[seg+x][int(q[x])-1]*duration/bw[0][seg+x]+duration,0)
        for y in xrange(0,l+1):
            ts[i][y]=max(br[seg+y][int(q[y])-1]*duration/bw[0][seg+y]-t[i+1][y],0)
        tst=np.sum(ts[i])
        ttt=duration*(l+1)+tst
        ps=tst/ttt
        delta=(t[i+1][l]-t[i+1][0])/(l+1)
### Calculate the probabilities
        if seg !=0:
            #p=A[int(Bw[0][seg-1])-1][int(Bw[0][seg])-1]*A[int(Bw[0][seg])-1][int(Bw[0][seg+1])-1]*A[int(Bw[0][seg+1])-1][int(Bw[0][seg+2])-1]*A[int(Bw[0][seg+2])-1][int(Bw[0][seg+3])-1]*A[int(Bw[0][seg+3])-1][int(Bw[0][seg+4])-1]
            #p=A[int(Bw[0][seg-1])-1][int(Bw[0][seg])-1]
            p=A[int(Bw[0][seg-1])-1][int(Bw[0][seg])-1]*A[int(Bw[0][seg])-1][int(Bw[0][seg+1])-1]*A[int(Bw[0][seg+1])-1][int(Bw[0][seg+2])-1]
        else:
            #p=A[2][int(Bw[0][seg])-1]*A[int(Bw[0][seg])-1][int(Bw[0][seg+1])-1]*A[int(Bw[0][seg+1])-1][int(Bw[0][seg+2])-1]*A[int(Bw[0][seg+2])-1][int(Bw[0][seg+3])-1]*A[int(Bw[0][seg+3])-1][int(Bw[0][seg+4])-1]
            #p=A[2][int(Bw[0][seg])-1]
            p=A[2][int(Bw[0][seg])-1]*A[int(Bw[0][seg])-1][int(Bw[0][seg+1])-1]*A[int(Bw[0][seg+1])-1][int(Bw[0][seg+2])-1]
        qoe[i][j]=p*(e-w1*vv-w2*ps+lamda*delta)
    qindex=np.argmax(qoe[i],axis=0)
    Q=Q+fi[qindex][:]
    t[i+1][0]=max(t[i][1]-br[seg][int(Q[seg])-1]*duration/bw[0][seg]+duration,0)
    for x in xrange(0,l):
        t[i+1][x+1]=max(t[i+1][x]-br[seg+x][int(Q[seg+x])-1]*duration/bw[0][seg+x]+duration,0)
```

```
        for y in xrange(0,l+1):
            ts[i][y]=max(br[seg+y][int(Q[seg+y])-1]*duration/bw[0][seg+y]-t[i+1][y],0)


### Calculate the final QoE value
Average=np.mean(Q)
print 'Average Quality', Average
varies=np.zeros((1,segnum-1))

for k in xrange(0,segnum-1):
    varies[0][k]=abs(Q[k+1]-Q[k])
Var=np.sum(varies)/(segnum-1)
print 'Quality Variation', Var


T=np.zeros((1,segnum+1))
Ts=np.zeros((1,segnum))

T[0][0]=5
for k in xrange(0,segnum):
    T[0][k+1]=max(T[0][k]-br[k][int(Q[k])-1]*duration/bw[0][k]+duration,0)
    Ts[0][k]=max(br[k][int(Q[k])-1]*duration/bw[0][k]-T[0][k],0)

TSS=np.sum(Ts)
TST=duration*segnum+TSS
PSS=TSS/TST
print 'Starvation Ratio', PSS
QoE=Average-w1*Var-w2*PSS
print 'QoE value', QoE

### Plot the figures
fig = plt.figure(1)
x=np.arange(0,segnum)
bit=np.zeros((1,segnum))
for i in xrange(0,segnum):
    bit[0][i]=br[i][int(Q[i])-1]
ax=fig.add_subplot(111)
Bitrate,=ax.plot(x,bit[0],label='Bitrate')
Bandwidth,=ax.plot(x,bw[0],label='Bandwidth')
plt.xlabel('Segment Index',fontsize=20)
plt.ylabel('Requested Bitrate and Bandwidth',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Requested Bitrate and Bandwidth for each segment',fontsize=30)
ax.legend()

plt.figure(2)
plt.plot(x,bit[0],label='Bitrate')
plt.xlabel('Segment Index',fontsize=20)
plt.ylabel('Requested Bitrate',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Requested Bitrate for each segment',fontsize=30)
plt.legend()


fig,left_axis=plt.subplots()

right_axis = left_axis.twinx()

p1, = left_axis.plot(x, bw[0],color='red',label='Bandwidth')
p2, = right_axis.plot(x,T[0][1:],color='blue',label='Buffer Reservation')

left_axis.set_xlabel('Segment Index',fontsize=20)
left_axis.set_ylabel('Bandwidth',fontsize=20)
right_axis.set_ylabel('Buffer Reservation',fontsize=20)

tkw = dict(size=5, width=1.5)
left_axis.tick_params(axis='y', colors=p1.get_color(), **tkw)
right_axis.tick_params(axis='y', colors=p2.get_color(), **tkw)
left_axis.tick_params(axis='x', **tkw)

left_axis.legend()


right_axis.legend(loc='upper left')


plt.show()
```

Appendix 7: ‘qlearning.py’

```
#import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from itertools import *
import os.path as path
import xlwt
import xlrd
import math



###  Search the segments path
filepath=path.abspath(path.dirname(__file__))

MEDIA_FILE          = 'dashtest'
VIDEO_DIR           = 'video'

segment_path=path.join(filepath,MEDIA_FILE,VIDEO_DIR)

### Open the bitrate and bandwdith file
bitratebook=xlrd.open_workbook(path.join(segment_path,'bigbuckbunny.xls'))
bandwidthbook=xlrd.open_workbook(path.join(segment_path,'bw.xls'))

bitratedata=bitratebook.sheets()[0]
bwdata=bandwidthbook.sheets()[0]

### Calculate the number of quality level and segment
qualitynum=bitratedata.ncols
segnum=bitratedata.nrows

### Write the data into lists
bitrate=[]
bw=[]

for rownum in xrange(0,segnum):
    for colnum in xrange(0,qualitynum):
        bitrate.append(bitratedata.cell(rowx=rownum,colx=colnum).value)
bitrate=[bitrate]

br=np.zeros((segnum,qualitynum))
for i in xrange(0,segnum):
    for j in xrange(0,qualitynum):
        br[i][j]=float(bitrate[0][i*qualitynum+j])
#print br

for rownum in xrange(0,segnum):
    bw.append(bwdata.cell(rowx=rownum,colx=1).value)
bw=[bw]
#print bw

### Future segments number
l=2

### Duration of each segment
duration=2

### Every states and actions
#list1=[1,2,3,4]
actions=list(product(xrange(1,5),repeat=l+1))
states=list(product(xrange(1,6),repeat=l+1))
for i in xrange(0,len(actions)):
    actions[i]=list(actions[i])
for i in xrange(0,len(states)):
    states[i]=list(states[i])

### Weights of QoE
w1=0.3333333333333333
w2=2
lamda=0.9

### Bandwidth level
bwlevel=[20,60,100,500,1000]
```

```
73
74  ### Define the QoE as reward function
75  def QoE(bandwidth, quality, bitrate, prebuffer):
76  ⟶v = []
77  ⟶T = []
78  ⟶Ts = []
79  ⟶l_qual = len(quality)
80  ⟶E = np.mean(quality)
81  ⟶for i in range(l_qual-1):
82  ⟶⟶v.append(abs(quality[i+1]-quality[i]))
83  ⟶V = np.mean(v)
84  ⟶T.append(prebuffer)
85  ⟶for i in range(l_qual):
86  ⟶⟶T.append(max(T[i] - bitrate[i]*2/bandwidth[i] + 2, 0))
87  ⟶⟶Ts.append(max(bitrate[i]*2/bandwidth[i] - T[i], 0))
88  ⟶if sum(Ts) == 0:
89  ⟶⟶Ts_thetafi = sum(Ts)
90  ⟶else:
91  ⟶⟶Ts_thetafi = sum(Ts).item()
92  ⟶Tt = 2*(l_qual) + Ts_thetafi
93  ⟶Ps = Ts_thetafi/Tt
94  ⟶QoE_prod = E-w1*V-w2*Ps
95  ⟶return QoE_prod, T[i]
96
97  ### The transition matrix
98
99  #A=[0.4  0.6  0  0  0
100 #   0.2  0.4  0.4  0  0
101 #   0  0.4  0.4  0.2  0
102 #   0  0  0.5  0.3  0.2
103 #   0  0  0  0.5  0.5]
104
105 ### Define the bandwidth prediction function
106 def bwpredict(pre):
107     B=np.zeros((1,l+1))
108     #print pre
109     if pre == 1:
110         B[0][0]=int(np.random.choice([1,2],1,p=[0.4,0.6]))
111     elif pre==2:
112         B[0][0]=int(np.random.choice([1,2,3],1,p=[0.2,0.4,0.4]))
113     elif pre==3:
114         B[0][0]=int(np.random.choice([2,3,4],1,p=[0.4,0.4,0.2]))
115     elif pre==4:
116         B[0][0]=int(np.random.choice([3,4,5],1,p=[0.5,0.3,0.2]))
117     elif pre==5:
118         B[0][0]=int(np.random.choice([4,5],1,p=[0.5,0.5]))
119     for i in xrange(0,l):
120         if B[0][i]==1:
121             B[0][i+1]=int(np.random.choice([1,2],1,p=[0.4,0.6]))
122         elif B[0][i]==2:
123             B[0][i+1]=int(np.random.choice([1,2,3],1,p=[0.2,0.4,0.4]))
124         elif B[0][i]==3:
125             B[0][i+1]=int(np.random.choice([2,3,4],1,p=[0.4,0.4,0.2]))
126         elif B[0][i]==4:
127             B[0][i+1]=int(np.random.choice([3,4,5],1,p=[0.5,0.3,0.2]))
128         elif B[0][i]==5:
129             B[0][i+1]=int(np.random.choice([4,5],1,p=[0.5,0.5]))
130     B=[int(B[0][0]),int(B[0][1]),int(B[0][2])]
131     return B
132     print B
133
134
135 ### Set the parameters in Q-learning
136 gamma = 0.99
137 alpha=0.01
138 prebuffer=5
139 Rmatrix = np.zeros((len(states), len(actions)))
140 Qmatrix=np.zeros((len(states),len(actions)))
141
142 steps = 0
143
144 BW=[]
```

```
Q=[]
start_state = bwpredict(3)
current_state=start_state
bandw=[]
q=[]
while steps < 1000:
    steps += 1
    BW=current_state
    bandw=bandw+BW
    actionindex=np.random.randint(0,len(actions))
    q=actions[actionindex]
    Q=Q+q
    next_state=bwpredict(current_state[1])
    futurereward=[]
    seg=steps%segnum
    i=states.index(next_state)
    r=np.zeros((1,len(actions)))
    for next_action in actions:
        j=actions.index(next_action)
        r[0][j]=Qmatrix[i][j]
    band=[]
    bitr=[]
    for x in xrange(0,len(BW)):
        k=x*steps%segnum
        band.append(bwlevel[int(BW[x])-1])
        bitr.append(br[k][int(q[x])-1])
    qstate,prebuffer=QoE(band,q,bitr,prebuffer)

    Qmatrix[states.index(current_state)][actionindex]=Qmatrix[states.index(current_sta
    te)][actionindex]+alpha*(qstate-Qmatrix[states.index(current_state)][actionindex])
    current_state=next_state
print Qmatrix

### Save the Q matrix as excel file
book = xlwt.Workbook()
sheet = book.add_sheet(u'sheet1',cell_overwrite_ok=True)

for i in xrange(0,len(states)):
    for j in xrange(0,len(actions)):
        sheet.write(i,j,Qmatrix[i][j])
        #sheet.write(i,0,i)
        #sheet.write(0,j,j)


book.save(path.join(segment_path,'Q.xls'))

#QOE=rewards(prebuffer,segnum,quality,bw)

bw[0] = [int(i) for i in bw[0]]

quality=[]

### Find the index in bandwdith level of predicted bandwdith
Bw=np.zeros((1,segnum))

for i in xrange(0,segnum):
    Bw[0][i]=int(bwlevel.index(bw[0][i])+1)

print Bw

### Decide the quality level according to the Q matrix
for i in xrange(0,segnum,1+1):
    bb=[int(Bw[0][i]),int(Bw[0][i+1]),int(Bw[0][i+2])]
    bwstate=states.index(bb)
    Qrow=Qmatrix[bwstate]
    if len(np.nonzero(Qrow)) != 0:
        Qcol=np.argmax(Qrow,axis=0)
    else:
        Qcol=len(actions)
    quality=quality+actions[Qcol]

print quality
```

```
215
216 ### Calculate the final QoE value
217 e=np.mean(quality)
218 print e
219 va=np.zeros((1,len(quality)-1))
220 for i in xrange(0,len(quality)-1):
221     va[0][i]=abs(quality[i+1]-quality[i])
222 Va=np.sum(va)/(len(quality)-1)
223 print Va
224 tt=np.zeros((1,len(quality)+1))
225 tt[0][0]=prebuffer
226 tst=np.zeros((1,len(quality)))
227 for j in xrange(0,len(quality)):
228     tt[0][j+1]=max(tt[0][j]-br[j][int(quality[j])-1]*duration/bw[0][j-1]+duration,0)
229     tst[0][j]=max(br[j][int(quality[j])-1]*duration/bw[0][j-1]-tt[0][j],0)
230 Tst=np.sum(tst)
231 Ttt=duration*len(quality)+Tst
232 Pst=Tst/Ttt
233 print Pst
234 QoE=e-w1*Va-w2*Pst
235 print QoE
236
237 ### Plot the figures
238 fig = plt.figure(1)
239 x=np.arange(0,segnum)
240 bit=np.zeros((1,segnum))
241 for i in xrange(0,segnum):
242     bit[0][i]=br[i][int(Q[i])-1]
243 ax=fig.add_subplot(111)
244 Bitrate,=ax.plot(x,bit[0],label='Bitrate')
245 Bandwidth,=ax.plot(x,bw[0],label='Bandwidth')
246 plt.xlabel('Segment Index',fontsize=20)
247 plt.ylabel('Requested Bitrate and Bandwidth',fontsize=20)
248 plt.tick_params(labelsize=15)
249 plt.title('Requested Bitrate and Bandwidth for each segment',fontsize=30)
250 ax.legend()
251
252 plt.figure(2)
253 plt.plot(x,bit[0],label='Bitrate')
254 plt.xlabel('Segment Index',fontsize=20)
255 plt.ylabel('Requested Bitrate',fontsize=20)
256 plt.tick_params(labelsize=15)
257 plt.title('Requested Bitrate for each segment',fontsize=30)
258 plt.legend()
259
260
261 fig,left_axis=plt.subplots()
262
263 right_axis = left_axis.twinx()
264
265 p1, = left_axis.plot(x, bw[0],color='red',label='Bandwidth')
266 p2, = right_axis.plot(x,T[0][1:],color='blue',label='Buffer Reservation')
267
268 left_axis.set_xlabel('Segment Index',fontsize=20)
269 left_axis.set_ylabel('Bandwidth',fontsize=20)
270 right_axis.set_ylabel('Buffer Reservation',fontsize=20)
271
272 tkw = dict(size=5, width=1.5)
273 left_axis.tick_params(axis='y', colors=p1.get_color(), **tkw)
274 right_axis.tick_params(axis='y', colors=p2.get_color(), **tkw)
275 left_axis.tick_params(axis='x', **tkw)
276
277 left_axis.legend()
278
279 right_axis.legend(loc='upper left')
280
281
282 plt.show()
```

Appendix 8: ‘qlearning2.py’

```
#import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from itertools import *
import os.path as path
import xlwt
import xlrd
import math


###  Search the segments path
filepath=path.abspath(path.dirname(__file__))

MEDIA_FILE        = 'dashtest'
VIDEO_DIR         = 'video'

segment_path=path.join(filepath,MEDIA_FILE,VIDEO_DIR)

### Open the bitrate and bandwdith file
bitratebook=xlrd.open_workbook(path.join(segment_path,'bigbuckbunny.xls'))
bandwidthbook=xlrd.open_workbook(path.join(segment_path,'bw.xls'))

bitratedata=bitratebook.sheets()[0]
bwdata=bandwidthbook.sheets()[0]

### Calculate the number of quality level and segment
qualitynum=bitratedata.ncols
segnum=bitratedata.nrows

### Write the data into lists
bitrate=[]
bw=[]

for rownum in xrange(0,segnum):
    for colnum in xrange(0,qualitynum):
        bitrate.append(bitratedata.cell(rowx=rownum,colx=colnum).value)
bitrate=[bitrate]

br=np.zeros((segnum,qualitynum))
for i in xrange(0,segnum):
    for j in xrange(0,qualitynum):
        br[i][j]=float(bitrate[0][i*qualitynum+j])
#print br

for rownum in xrange(0,segnum):
    bw.append(bwdata.cell(rowx=rownum,colx=1).value)
bw=[bw]
#print bw

### Future segments number
l=2

### Duration of each segment
duration=2

### Every states and actions
#list1=[1,2,3,4]
actions=list(product(xrange(1,5),repeat=l+1))
states=list(product(xrange(1,6),repeat=l+1))
for i in xrange(0,len(actions)):
    actions[i]=list(actions[i])
for i in xrange(0,len(states)):
    states[i]=list(states[i])

### Weights of QoE
w1=0.3333333333333333
w2=2
lamda=0.9

### Bandwidth level
bwlevel=[20,60,100,500,1000]

```

```
### Define the QoE as reward function
def QoE(bandwidth, quality, bitrate, prebuffer):
    v = []
    T = []
    Ts = []
    l_qual = len(quality)
    E = np.mean(quality)
    for i in range(l_qual-1):
        v.append(abs(quality[i+1]-quality[i]))
    V = np.mean(v)
    T.append(prebuffer)
    for i in range(l_qual):
        T.append(max(T[i] - bitrate[i]*2/bandwidth[i] + 2, 0))
        Ts.append(max(bitrate[i]*2/bandwidth[i] - T[i], 0))
    if sum(Ts) == 0:
        Ts_thetafi = sum(Ts)
    else:
        Ts_thetafi = sum(Ts).item()
    Tt = 2*(l_qual) + Ts_thetafi
    Ps = Ts_thetafi/Tt
    QoE_prod = E-w1*V-w2*Ps
    return QoE_prod, T[i]

### The transition matrix

#A=[0.4  0.6   0    0    0
#   0.2  0.4  0.4   0    0
#    0   0.4  0.4  0.2   0
#    0    0   0.5  0.3  0.2
#    0    0    0   0.5  0.5]

### Define the bandwidth prediction function
def bwpredict(pre):
    B=np.zeros((1,l+1))
    #print pre
    if pre == 1:
        B[0][0]=int(np.random.choice([1,2],1,p=[0.4,0.6]))
    elif pre==2:
        B[0][0]=int(np.random.choice([1,2,3],1,p=[0.2,0.4,0.4]))
    elif pre==3:
        B[0][0]=int(np.random.choice([2,3,4],1,p=[0.4,0.4,0.2]))
    elif pre==4:
        B[0][0]=int(np.random.choice([3,4,5],1,p=[0.5,0.3,0.2]))
    elif pre==5:
        B[0][0]=int(np.random.choice([4,5],1,p=[0.5,0.5]))
    for i in xrange(0,1):
        if B[0][i]==1:
            B[0][i+1]=int(np.random.choice([1,2],1,p=[0.4,0.6]))
        elif B[0][i]==2:
            B[0][i+1]=int(np.random.choice([1,2,3],1,p=[0.2,0.4,0.4]))
        elif B[0][i]==3:
            B[0][i+1]=int(np.random.choice([2,3,4],1,p=[0.4,0.4,0.2]))
        elif B[0][i]==4:
            B[0][i+1]=int(np.random.choice([3,4,5],1,p=[0.5,0.3,0.2]))
        elif B[0][i]==5:
            B[0][i+1]=int(np.random.choice([4,5],1,p=[0.5,0.5]))
    B=[int(B[0][0]),int(B[0][1]),int(B[0][2])]
    return B
    print B


### Set the parameters in Q-learning
gamma = 0.99
alpha=0.01
prebuffer=5

segindex=segnum/(l+1)
Rmatrix = np.zeros((len(states), len(actions)))
Qmatrix=np.zeros((segindex,len(states),len(actions)))

for k in xrange(0,segindex):
    BW=[]
```

```
    Q=[]
    bandw=[]
    q=[]
    steps = 0
    current_state=states[np.random.randint(0,len(states))]
    while steps < 1000:
        steps += 1
        BW=current_state
        bandw=bandw+BW
        actionindex=np.random.randint(0,len(actions))
        q=actions[actionindex]
        Q=Q+q
        next_state=bwpredict(current_state[1])
        futurereward=[]
        seg=steps%segnum
        i=states.index(next_state)
        r=np.zeros((1,len(actions)))
        for next_action in actions:
            j=actions.index(next_action)
            r[0][j]=Qmatrix[k][i][j]
        band=[]
        bitr=[]
        for x in xrange(0,len(BW)):
            band.append(bwlevel[int(BW[x])-1])
            bitr.append(br[k+x][int(q[x])-1])
        qstate,prebuffer=QoE(band,q,bitr,prebuffer)

        Qmatrix[k][states.index(current_state)][actionindex]=Qmatrix[k][states.index(c
        urrent_state)][actionindex]+alpha*(qstate-Qmatrix[k][states.index(current_stat
        e)][actionindex])
        current_state=next_state

print Qmatrix[26]

### Save the Q matrix as excel file
book = xlwt.Workbook()
sheet = book.add_sheet(u'sheet1',cell_overwrite_ok=True)

for i in xrange(0,len(states)):
    for j in xrange(0,len(actions)):
        sheet.write(i,j,Qmatrix[26][i][j])
        #sheet.write(i,0,i)
        #sheet.write(0,j,j)


book.save(path.join(segment_path,'Q26.xls'))

### Find the index in bandwdith level of predicted bandwdith
bw[0] = [int(i) for i in bw[0]]


Bw=np.zeros((1,segnum))
#print bw
for i in xrange(0,segnum):
    Bw[0][i]=int(bwlevel.index(bw[0][i])+1)

### Decide the quality level according to the Q matrix
quality=[]
for i in xrange(0,segindex):
    bb=[int(Bw[0][i]),int(Bw[0][i+1]),int(Bw[0][i+2])]
    bwstate=states.index(bb)
    Qrow=Qmatrix[i][bwstate]
    if len(np.nonzero(Qrow)) != 0:
        Qcol=np.argmax(Qrow,axis=0)
    else:
        Qcol=len(actions)
    quality=quality+actions[Qcol]

print quality


### Calculate the final QoE value
```

```
e=np.mean(quality)
print e
va=np.zeros((1,len(quality)-1))
for i in xrange(0,len(quality)-1):
    va[0][i]=abs(quality[i+1]-quality[i])
Va=np.sum(va)/(len(quality)-1)
print Va
tt=np.zeros((1,len(quality)+1))
tt[0][0]=prebuffer
tst=np.zeros((1,len(quality)))
for j in xrange(0,len(quality)):
    tt[0][j+1]=max(tt[0][j]-br[j][int(quality[j])-1]*duration/bw[0][j-1]+duration,0)
    tst[0][j]=max(br[j][int(quality[j])-1]*duration/bw[0][j-1]-tt[0][j],0)
Tst=np.sum(tst)
Ttt=duration*len(quality)+Tst
Pst=Tst/Ttt
print Pst
QoE=e-w1*Va-w2*Pst
print QoE

### Plot the figures
fig = plt.figure(1)
x=np.arange(0,segnum)
bit=np.zeros((1,segnum))
for i in xrange(0,segnum):
    bit[0][i]=br[i][int(Q[i])-1]
ax=fig.add_subplot(111)
Bitrate,=ax.plot(x,bit[0],label='Bitrate')
Bandwidth,=ax.plot(x,bw[0],label='Bandwidth')
plt.xlabel('Segment Index',fontsize=20)
plt.ylabel('Requested Bitrate and Bandwidth',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Requested Bitrate and Bandwidth for each segment',fontsize=30)
ax.legend()

plt.figure(2)
plt.plot(x,bit[0],label='Bitrate')
plt.xlabel('Segment Index',fontsize=20)
plt.ylabel('Requested Bitrate',fontsize=20)
plt.tick_params(labelsize=15)
plt.title('Requested Bitrate for each segment',fontsize=30)
plt.legend()


fig,left_axis=plt.subplots()

right_axis = left_axis.twinx()

p1, = left_axis.plot(x, bw[0],color='red',label='Bandwidth')
p2, = right_axis.plot(x,T[0][1:],color='blue',label='Buffer Reservation')

left_axis.set_xlabel('Segment Index',fontsize=20)
left_axis.set_ylabel('Bandwidth',fontsize=20)
right_axis.set_ylabel('Buffer Reservation',fontsize=20)

tkw = dict(size=5, width=1.5)
left_axis.tick_params(axis='y', colors=p1.get_color(), **tkw)
right_axis.tick_params(axis='y', colors=p2.get_color(), **tkw)
left_axis.tick_params(axis='x', **tkw)

left_axis.legend()

right_axis.legend(loc='upper left')


plt.show()
```

Appendix 9: ‘Q.py’

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import torch.optim as optim
from random import random
import pandas
import matplotlib.pyplot as plt

######################## NN define
# Defining the class Net which is used
# for creating the neural network
class Net(nn.Module):
    # Initialization function
    def __init__(self):
        super(Net, self).__init__()
        # Defining the network layers
        self.affine1 = nn.Linear(5, 256)
        self.affine2 = nn.Linear(256, 4)
        # Defining network parameters
        self.bw = []
        self.P_bw = np.zeros((5, 7))
        self.actions = []
        self.br = []
        self.QoE = []
    # Defining forward propagation pass
    # Setting up how layers are connected
    # and which activation functions
    # are used for each layer
    def forward(self, x):
        x = torch.sigmoid(self.affine1(x))
        x = torch.sigmoid(self.affine2(x))
        # x is the oputput of the network
        return x
        
        
# Creating the neural network
# Create one object of the class Net
net = Net()
# Optimizer is network optimization function
optimizer = optim.Adam(net.parameters(), lr = 0.0000001)
# Criterion defines how network loss is calculated
# based on which backward propagation is prerformed
criterion = nn.MSELoss()
# Initial values for transition probability matrix
net.P_bw[0, 1] = 0.4
net.P_bw[0, 2] = 0.6
net.P_bw[1, 1] = 0.2
net.P_bw[1, 2] = 0.4
net.P_bw[1, 3] = 0.4
net.P_bw[2, 2] = 0.4
net.P_bw[2, 3] = 0.4
net.P_bw[2, 4] = 0.2
net.P_bw[3, 3] = 0.5
net.P_bw[3, 4] = 0.3
net.P_bw[3, 5] = 0.2
net.P_bw[4, 4] = 0.5
net.P_bw[4, 5] = 0.5

##############################################
# Function definitions

# Function used to extract current bitrates
# for l+1 segments from the whole video
def current_states(br, l, t):
    states = torch.zeros(l+1, 4)
    for i in range(l+1):
        for j in range(4):
            states[i, j] = br[t+i, j]
    return states

# Function which returns current input
```

```
# for the neural network from
# l+1 segments
def current_input(states, bw):
    n_input = []
    for i in range(4):
        n_input.append(states[i].item())
    n_input.append(bw)
    return torch.tensor(n_input)

# Function used to calculate rewards
# for current input
def reward(n_input):
    reward = []
    for i in range(4):
        temp = n_input[4]-n_input[i]
        if temp<0:
            temp = 10000
        reward.append(0.01*temp)
    return torch.tensor(reward)

# Function which predicts current bandwidth values
# and updates transition probability matrix
def bw_predict(bw):
    l_buff = len(net.bw)
    bw_curr = net.bw[l_buff-1]
    for i in range(7):
        if bw_curr == bw[i]:
            ind = i
    bw_pred = int(np.random.choice(bw[(ind-1):(ind+2)], 1, p = net.P_bw[ind-1,
        (ind-1):(ind+2)]))
    return bw_pred

# Function used to calculate possible length, l
# from the whole video length
def choose_l(vid_len):
    l_seg = []
    for i in range(vid_len+1):
        if vid_len%(i+1) == 0:
            l_seg.append(i+1)
    return l_seg

### Tian Tian's code, slightly modified
# This function is used to calculate QoE values
def QoE(bandwidth, quality, bitrate, prebuffer):
    v = []
    T = []
    Ts = []
    l_qual = len(quality)
    E = np.mean(quality)
    for i in range(l_qual-1):
        v.append(abs(quality[i+1]-quality[i]))
    V = np.mean(v)
    T.append(prebuffer)
    for i in range(l_qual):
        T.append(max(T[i] - bitrate[i]*2/bandwidth[i] + 2, 0))
        Ts.append(max(bitrate[i]*2/bandwidth[i] - T[i], 0))
    if sum(Ts) == 0:
        Ts_thetafi = sum(Ts)
    else:
        Ts_thetafi = sum(Ts).item()
    Tt = 2*(l_qual) + Ts_thetafi
    Ps = Ts_thetafi/Tt
    QoE_prod = E-w1*V-w2*Ps
    return QoE_prod, T[i], T
###
########### Training loop

# Setting up the possible bandwidth values
# Note: zeros at the beginning and the end
#       are used as zero padding, and will
#       never be chosen as bandwidth value
bw = [0, 20,60,100,500,1000, 0]
```

```
144
145 # Two videos are available for the training
146 # and user may choose which one to use
147 # For this part, input is Excel file, which contains table
148 # of bitrate values for each quality level
149 # These files were provided by coworker on this project
150 # and used without changes
151 vid = input("Choose training video: bunny bear --> ")
152 if vid == 'bunny':
153 ----training_video = pandas.read_excel('bigbuckbunny.xls', 0)
154 else:
155 ----training_video = pandas.read_excel('bear.xls', 0)
156
157 # Reading Excel file table to a matrix
158 # and gaining the video length, i.e.
159 # total number of segments
160 train_vid = training_video.as_matrix()
161 s = np.shape(train_vid)
162 train_vid_len = s[0]
163
164 # Acquiring all possible values for l
165 # from which the user chooses one for the training
166 l = choose_l(train_vid_len)
167 print(l)
168 l = int(input("Choose one of the values for l: "))
169 l = l-1
170
171 # Offering the user possible number of episodes
172 # for training process
173 # Note: These values are optional, and user may choose
174 #       arbitrary number of episodes for the training
175 num_ep = [100, 200, 300, 500, 1000]
176 print(num_ep)
177 num_ep = int(input("Choose number of episodes for training: "))
178
179 ### Setting up the training process parameters ###
180 # eps is the probability used in Epsilon-greedy
181 # policy of choosing whether the random action
182 # or best action - eploration v exploitation
183 # In this case, it is used changed value for
184 # Epsilon, it starts with large value, i.e.
185 # 0.99 (this is initialized further in the algorithm)
186 # and it is decreased with eps_dec value, until
187 # it reaches eps_min value
188 eps_min = 0.1
189 eps_dec = 0.001
190
191 # Gamma is used to calculate the referent Q(s, a) value
192 gamma = 0.99
193
194 # w1 and w2 are used to calculate the QoE value
195 w1 = 0.333333333333
196 w2 = 2
197
198 ### Main training loop ###
199 for nep in range(num_ep):
200 ----# Initializing first bandwidth value
201 ----first = np.random.randint(1, 5)
202 ----bw_curr = bw[first]
203 ----net.bw.append(bw_curr)
204 ----# Initializing first prebuffer size
205 ----prebuffer = 10
206 ----# Initializing probability value Epsilon
207 ----eps = 0.99
208 ----for segment in range(train_vid_len):
209 --------if segment%(l+1) == 0:
210 ------------# Acquiring current l+1 states bitrate values
211 ------------states = current_states(train_vid, l, segment)
212 ------------for state in range(l):
213 ----------------# Acquiring current input
214 ----------------n_input = current_input(states[state, :], bw_curr)
215 ----------------# Acquiring currend rewards
```

```
                    rewards = reward(n_input)
                    # Calculating output values for current input
                    # Forward propagation
                    # This produces Q(s) values
                    Q_s = net(n_input)
                    # Choosing random value uniformly
                    # This value is used for Epsilon-greedy policy
                    e = random()
                    # Choosing the action according to the
                    # Epsilon-greedy policy
                    if e < eps:
                        action = np.random.randint(0, 3)
                    else:
                        action = torch.argmax(Q_s).item()
                    # Decreasing the Epsilon value
                    if eps > eps_min:
                        eps -= eps_dec
                    else:
                        eps = eps_min
                    # After choosing the action
                    # Q(s, a) value is acquired
                    Q_s_a = Q_s[action]
                    # Calculating Q(s', a) value -
                    # Max Q-value in next state, s'
                    next_input = current_input(states[state+1, :], bw_curr)
                    next_Q_s_a = torch.max(net(next_input))
                    # Calculating reference Q(s, a) value
                    # which is compared to the current Q(s, a) value
                    # in order to calculate current loss
                    loss = criterion(Q_s_a, rewards[action] + gamma*next_Q_s_a.detach())
                    # Performing backward propagation
                    # and updating neural network weights
                    optimizer.zero_grad()
                    loss.backward()
                    optimizer.step()
                    # Predicting next bandwidth value
                    bw_curr = bw_predict(bw)
                    net.bw.append(bw_curr)
                    # Saving chosen action and bitrate value
                    # which is further used to calculate
                    # internal QoE value
                    net.br.append(n_input[action])
                    net.actions.append(action+1)
                # After predicting the quality level for
                # l+1 segments, internal QoE value is calculated
                QoE_curr, prebuffer, T = QoE(net.bw, net.actions, net.br, prebuffer)
                net.QoE.append(QoE_curr)
                del net.bw[:]
                del net.br[:]
                del net.actions[:]
                net.bw.append(bw_curr)
            if nep%10 == 0:
                print('Episode number: ', nep+10)
                print('Loss: ', loss.item())
                print('Q(s): ', Q_s)
                print('Input: ', n_input)
                print('Action: ', action+1)
                if e < eps:
                    print('Random action')
                print('QoE: ', QoE_curr)

                print('################################################################################
                ###################')
            del net.QoE[:]


############## Testing loop

# One video is used for testing process
# Bitrate values are acquired from the
# Excel file, provided by coworker on the project
```

```
testing_video = pandas.read_excel('bigbuckbunny.xls', 0)
test_vid = testing_video.as_matrix()
s = np.shape(test_vid)
test_vid_len = s[0]

# Choosing the number of segments, l+1
# for testing video
l = choose_l(test_vid_len)
print(l)
l = int(input("Choose l: "))
l = l-1

### Main testing loop ###
# Testing loop is similar to the training loop
# except from updating the neural network
b = np.random.randint(1, 5)
bw_t = bw[b]
prebuff = 5

bw_all = []
br_all = []
ac_all = []
bw_all.append(bw_t)


for segment in range(test_vid_len):
    if segment%(l+1) == 0:
        states = current_states(test_vid, l, segment)
        for state in range(l+1):
            inn = current_input(states[state, :], bw_t)
            Q = net(inn)
            net.bw.append(bw_t)
            action = torch.argmax(Q).item()
            net.actions.append(action+1)
            net.br.append(inn[action])
            bw_t = bw_predict(bw)
            bw_all.append(bw_t)
            br_all.append(inn[action])
            ac_all.append(action+1)
        QoE_t, prebuff, T = QoE(net.bw, net.actions, net.br, prebuff)
        net.QoE.append(QoE_t)
        print('Predicted Q(s, a) values: ', Q)
        print('Optimal quality: ', torch.argmax(Q).item()+1)
        print('Input: ', inn)
        print('Bitrate-bandwidth pair: ', inn[torch.argmax(Q)].item(), ', ',
              inn[4].item())
        print('QoE: ', QoE_t)
        print('#############################################################################
              #######')
        del net.bw[:]
        del net.actions[:]
        del net.br[:]
        net.bw.append(bw_t)
# Finaly, average QoE value is calculated and printed
print('Final QoE: ', np.mean(net.QoE))

# Plot the results
QoE_t, prebuff, T = QoE(bw_all, ac_all, br_all, 10)
fig, ax = plt.subplots()
ax.plot(br_all, label="Bitrate")
ax.plot(bw_all, label="Bandwidth")
ax.set_xlabel("Segment number")
ax.tick_params(labelsize=15)
plt.title('Requested Bitrate and Bandwidth for each segment',fontsize=30)
ax.legend()
plt.show()
fig, ax = plt.subplots()
ax.plot(T)
ax.set_xlabel('Segment number')
ax.set_ylabel('Buffer size [s]')
```

```
355  plt.show()
356
357  fig, ax = plt.subplots()
358  ax.plot(br_all, label="Bitrate")
359  #ax.plot(bw_all, label="Bandwidth")
360  ax.set_xlabel("Segment number")
361  ax.legend()
362  plt.show()
363
364  fig,left_axis=plt.subplots()
365  #fig.subplots_adjust(right_axis=0,75)
366
367  right_axis = left_axis.twinx()
368
369  p1, = left_axis.plot(bw_all,color='red',label='Bandwidth')
370  p2, = right_axis.plot(T,color='blue',label='Buffer Reservation')
371
372
373
374  left_axis.set_xlabel('Segment Index',fontsize=20)
375  left_axis.set_ylabel('Bandwidth',fontsize=20)
376  right_axis.set_ylabel('Buffer Reservation',fontsize=20)
377
378  #left_axis.yaxis.label.set_color(p1.get_color())
379  #right_axis.yaxis.label.set_color(p2.get_color())
380
381  tkw = dict(size=5, width=1.5)
382  left_axis.tick_params(axis='y', colors=p1.get_color(), **tkw)
383  right_axis.tick_params(axis='y', colors=p2.get_color(), **tkw)
384  left_axis.tick_params(axis='x', **tkw)
385
386  left_axis.legend()
387
388  right_axis.legend(loc='upper left')
389  #plt.show()
390
391
392  plt.show()
393
394
395
396
397
398
399
```