Signal Processing: *Image Communication* 39 (2015) 305–315

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Dynamic rate adaptation for adaptive video streaming in wireless networks ☆

Siyuan Xiang [a], Min Xing [a], Lin Cai [a], Jianping Pan [b]

[a] *Department of Electrical & Computer Engineering, University of Victoria, Victoria, BC, Canada*
[b] *Department of Computer Science, University of Victoria, Victoria, BC, Canada*

ARTICLE INFO

Article history:
Received 8 April 2015
Received in revised form
26 June 2015
Accepted 14 August 2015
Available online 28 September 2015

Keywords:
DASH
Wireless multimedia

ABSTRACT

In this paper, we investigate the streaming strategy for dynamic adaptive streaming over HTTP (DASH). Specifically, we focus on the rate adaptation algorithm for streaming scalable video (H.264/SVC) in wireless networks. We model the rate adaptation problem as a Markov Decision Process (MDP), aiming to find an optimal streaming strategy in terms of user-perceived quality of services (QoS) such as playback interruption, average playback quality and playback smoothness. We then obtain the optimal MDP solution using dynamic programming. However, the optimal solution requires the knowledge of the available bandwidth statistics and has a large number of states, which makes it difficult to obtain the optimal solution in real time. Therefore, we further propose an online algorithm which integrates the learning and planning process. The proposed online algorithm collects bandwidth statistics and makes streaming decisions in real time. A reward parameter has been defined in our proposed streaming strategy, which can be adjusted to make a good trade-off between the average playback quality and playback smoothness. We also use a simple testbed to validate our proposed algorithm. Experimental results show the feasibility of the proposed algorithm and its advantage over the existing work.

1. Introduction

Current statistics show that video applications account for the highest percentage of the traffic mix in the Internet. Cisco forecasts that, by 2018, the sum of all forms of video, including TV, video on demand, Internet video, and peer-to-peer (P2P) video, will account for 79% of the global consumer network traffic, while mobile video will account for more than 75% of the total mobile network traffic [2].

The increasingly popular video websites such as YouTube and Vimeo will be the major providers of mobile videos. Progressive download is currently the dominant video delivery techniques of these video websites. It has several advantages over the traditional streaming techniques using RTP/UDP. First, it is simple to deploy. At the server side, any web server can host videos and serve as a streaming server; at the client side, the user only needs a flash player or web browser supporting HTML5 for video playback. Second, the HTTP/TCP protocols used in progressive download are more firewall and network address translation (NAT) friendly, and the congestion control mechanism in TCP simplifies the design of the application layer. Third, for progressive download, a server can store several versions of a video to meet the requirements of heterogeneous users, so a user can select the right version of the video according to the device decoding capability, display size and available network bandwidth.

☆ Preliminary results of this paper have been presented at ACM MMSys'12 [1]. New technical contributions, including the design of the online algorithm, the video segment storage structure, the SVC video player implementation and the experiments using the wireless testbed, are presented in Sections 4 and 5, respectively.

http://dx.doi.org/10.1016/j.image.2015.08.007
0923-5965/

However, selecting the appropriate version of a video to match the available bandwidth may not be easy for users and their decisions might be error-prone. In addition, with progressive download, the client always downloads as much video data as possible. Plissonneau and Biersack [3] report that only half of the videos are fully downloaded and this number drops dramatically when the users are not satisfied with the video quality. It is likely that when a user turns off the video player or switches to another video, a large amount of un-watched video has been buffered unnecessarily, which wastes the resources of both the network and the end-systems. It is particularly undesirable for mobile devices with limited energy supply.

Dynamic adaptive streaming over HTTP (DASH) [4] is a promising technique to overcome the aforementioned disadvantages of progressive download. Videos encoded in different versions are chopped into small segments. After the client receives one segment, it has a chance to decide which version of the video to request for the next segment, based on the current network condition. Thus, rate adaptation can be performed at the client side naturally and flexibly. Also, the client has a chance to control the client-side queue length to avoid streaming buffer overflow, e.g., when the download rate is much higher than the playback rate.

Currently, commercial adaptive streaming products such as Microsoft Smooth Streaming and Apple Live Streaming support single-layer H.264 advanced video coding (AVC) encoded videos. Multiple versions of a video with different resolution, frame rate and quality are obtained by encoding the source video multiple times with different configurations, and the different versions of the video are completely independent of each other. Thus, not only more server storage space is needed, but also the web caching hit-ratio is reduced.

Recently, scalable video coding (H.264/SVC) has been introduced to the DASH framework to improve the system performance [5]. With SVC, a video is encoded once only, and it can be decoded many times with different resolution, frame rate and quality. However, how to improve the rate adaptation algorithm to provide users with a satisfactory quality of services (QoS) is still a challenging and open question. The problem is even more challenging when a user uses a handheld device via a wireless access link for video streaming, as the handheld devices typically have limited energy supply and computation capacity, and the wireless links are highly dynamic due to the time-varying fading, shadowing, interference and hand-off, all of which motivated our work.

In this paper, we design the rate adaptation algorithm for streaming scalable video over HTTP in wireless networks. The main contributions of this paper are threefold. First, we formulate the rate adaptation problem as a finite Markov Decision Process (MDP), aiming to find an optimal streaming strategy in terms of user-perceived QoS such as playback interruption, average playback quality and playback smoothness. We obtain the optimal streaming strategy by dynamic programming under the reinforcement learning framework [6]. A reward parameter is defined in our proposed strategy, which can be adjusted to make a trade-off between average playback quality and smoothness. Second, since the optimal solution requires the knowledge of available bandwidth statistics and has a high computational complexity, which makes it difficult to obtain the optimal solution in real time, we propose an online algorithm which integrates the learning and planning process, i.e., the proposed algorithm collects bandwidth statistics and makes streaming decisions in real time. Third, we have prototyped a scalable video streaming framework including the server-side video pre-processing and client-side SVC video playing back. A real sample video encoded in SVC is used to evaluate the proposed streaming strategies and compare them with the existing work using both wireless testbed experiments and simulations. The experimental and simulation results show the advantage of the proposed algorithms in practical settings.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 formulates the optimal streaming problem as an MDP. Section 4 presents the proposed optimal streaming policy and the online algorithm. The evaluation framework, testbed configurations and experimental results are described and given in Section 5, followed by concluding remarks and further research issues in Section 6.

2. Background and related work

Different from the application-layer multicasting [7], in a DASH system, rate adaptation is conducted at the client side, which is also called pull-based rate adaptation [8]. At the server side, a source video is encoded into different versions with different resolution, frame rate and quality. For each version, the video is divided into small segments. A web server can host these segments and send them to the clients upon HTTP requests. At the client side, after a user clicks the play button, the streaming starts. The video player first obtains the general information of the video, such as the number of different versions and the corresponding resolution, frame rate and quality of each version. Then, the video player will decide the right version according to its own display size, decoding capability and network condition. Usually, the playback does not start until a sufficient number of segments are received. After the client receives a segment completely, the rate adaptation algorithm will decide which version to request for the next segment based on the current network condition and the client-side state such as the number of buffered segments. In this way, the workload of the server is reduced dramatically. Fig. 1 shows the general workflow of the video player in a DASH-like system.

There are extensive research efforts on the adaptive video streaming over HTTP [4,9,10,11]. Stockhammer [4] introduced the 3GPP specification of the dynamic adaptive streaming over HTTP, which describes the framework of the adaptive streaming system. In [10], the commercial adaptive streaming products including Microsoft Smooth Streaming and Netflix player and the open source media framework (OSMF) player were evaluated and compared. The results show that the performance of these products still needs to be improved substantially.

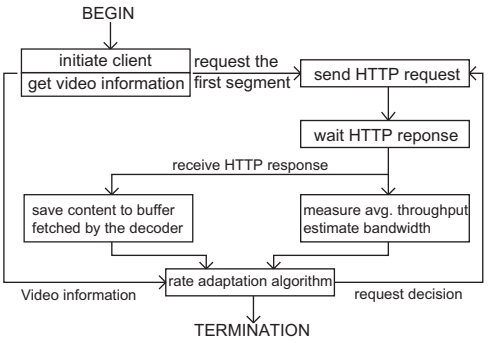

Fig. 1. Video player state diagram.

Liu et al. proposed a rate adaptation algorithm for adaptive video streaming [9]. The decision of switching to a video version of a higher or lower bit-rate is made based on the measured segment fetch time, which can be converted to the average segment throughput and buffer state. The algorithm is evaluated using constant bit-rate (CBR), single-layer video traffic only, and the queue length may sometimes exceed the maximum buffer size. In [11], a quality-adaptation controller based on the feedback control theory was proposed. The controller tries to maintain the buffer level as stable as possible to match the video bit-rate with the available bandwidth. As the server needs to maintain the information for each user to perform rate adaptation, the complexity of the server is increased. To improve the performance of adaptive video streaming over HTTP for mobile users, the authors in [12] tried to reduce the computation load of MDP approach. They devised both off-line algorithm based on global bandwidth statistics and on-line algorithm to calculate the optimal solution periodically. In [13], Yan et al. proposed an evidence theory-based admission control scheme in cellular networks to guarantee the QoE continuum for both existing and new users. They derived the admission decision logic by combining the weighted evidence of all users, to address the issues of uncertainty and inaccuracy of QoE management.

Recently, SVC has been introduced to adaptive video streaming. With SVC, we can encode video once and decode the bitstream multiple times with different resolution, frame rate and video quality [14,15], so the encoding time and server storage space can be saved, which is particularly important for live or high-quality stored video streaming. In addition, thanks to the layered structure of SVC, we may even upgrade an already received segment to a higher quality in certain conditions [16,17]. Sánchez et al. [5] showed the advantage of using SVC in adaptive HTTP streaming over the single-layer AVC in terms of caching efficiency. In [16], the authors proposed a priority-based media delivery strategy using SVC with RTP and HTTP streaming. In the pre-buffering phase, the most important base layer is transmitted first, so there are more base-layer frames than enhancement-layer frames in the buffer. This scheme was designed assuming that the temporary bandwidth reduction is the only possible bandwidth variation, and the bandwidth will restore to a normal level after the temporary reduction. Thus, it cannot fully handle the random variation of the available bandwidth at the bottleneck in wireless networks. In [17], how to use multiple links to improve video quality considering the different costs of various links was investigated. To investigate the adaptive video streaming over whitespace fast-fading channel, the authors in [18] proposed a new MDP-based algorithm to take the advantage of the flexibility of SVC video.

Different from the existing approaches, in this paper, we focus on the rate adaptation algorithm for streaming SVC video in wireless networks, considering the random and less predictable variation of the available bandwidth in wireless access links. We also consider the more general case where the layered video is encoded in variable bit-rate (VBR). The fairness of the proposed algorithm in multiple clients scenario has been investigated. And extensive experiments using realistic bandwidth and video traces have been conducted.

3. Problem formulation

Considering the limited computation capacity of handheld devices and the high variation of wireless access links, we formulate the optimal rate adaptation problem as a finite Markov Decision Process, which can deal with the random network condition with a relatively simple approach. For each video segment, the client uses MDP to make a decision on which action to conduct given the current client state. There are four components for MDP, i.e., action, state, transition probability and reward. MDP or reinforcement learning has also been successfully used in the medical imaging area [19].

As shown in Fig. 1, after a segment is obtained completely, the rate adaptation algorithm has a chance to decide the video version of the next segment to request and whether the client should be idle for a while to avoid streaming buffer overflow. We define the sequential actions as $\{a_t\}, t=0,1,\ldots$. a_t is the decision made at step t, where the step duration equals the time to retrieve one segment. Note that the step duration is not a constant, since the segment download time varies according to the segment size and the available bandwidth. L is the number of versions. The action set for a given state is $\mathcal{A}(s)=\{A_i, A_u, A_w\}$, where A_i $(i=-L+1,\ldots,L-1)$ means to request the next segment with i layer higher (for $i\geq 0$) or lower (for $i<0$) than the current one, A_u means to "upgrade" the last received segment to a higher quality version, and A_w means to wait for a time duration of T_s which is the constant playback time of a segment.

We define a state at step t as $s_t=(q_t, \Delta q_t, v_t, \Delta v_t, bw_t, d_t)$. Here, q_t is the queue length in terms of the number of buffered frames. Obviously, q_t is in the range of $(0, F)$, where $F=B_T\times N_s$, B_T is the target buffer size in terms of the number of segments, and N_s is the number of frames per segment. Δq_t is the queue length variation after a new segment has been retrieved, i.e., $\Delta q_t=q_t-q_{t-1}$, which indicates whether the requested video's bit-rate matches the available bandwidth. Δq_t is in the range of $[-F, N_s]$. v_t is the version index of the last received segment. Δv_t indicates the difference of video versions requested in consecutive

steps. bw_t is the available bandwidth at step t. d_t is the number of received segments, which is in the range of $[0, N_T]$, where N_T is the total number of segments the client needs to request.

From the definition of the states, we can observe that the Markov property exists, since all of these states depend on their immediately previous state only, i.e.,

$$\Pr\{s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0\} = \Pr\{s_{t+1}|s_t, a_t\}. \quad (1)$$

To obtain the state transition probability, the most challenging issue is to obtain the model for bw_t. For wireless streaming scenarios, the bottleneck is often in the wireless access link due to contention or mobility, and the finite-state Markov chain has been widely used to model the variation of wireless channels [20,21]. Thus, we use a discrete-time finite-state Markov model to capture the variation of the bandwidth, and the state transition probabilities can be obtained from the measurement or derived from the wireless channel model [20]. Given the time duration for downloading the current segment, we can estimate the probability distribution of the bandwidth for the next segment using the state transition probability matrix of the Markov model.

For the problem of our interest, we can derive the state transition probability for the MDP by

$$\mathcal{P}^a_{ss'} = \Pr\{s_{t+1} = s'|s_t = s, a_t = a\}. \quad (2)$$

The state at step t is $s = (q, \Delta q, v, \Delta v, bw, d)$. If action $a_t = A_i$ is selected, with probability $\mathcal{P}^a_{ss'} = \Pr\{bw'|bw\}$, the new state will be $s' = (q', \Delta q', v', \Delta v', bw', d')$, i.e.,

$$\begin{aligned} &v' = v + i, \quad \Delta v' = i, \\ &q' = q - \lceil (m^{v'}_{d+1} \times f)/bw' \rceil + N_s, \\ &\Delta q' = q' - q, \quad d' = d + 1, \end{aligned} \quad (3)$$

where $m^{v'}_{d+1}$ is the size of segment $d+1$ in version v' and f is the playback frame rate (since we are dealing with the stored video streaming, the client can have the knowledge of the size of every segment). If $a_t = A_u$, the new state is

$$\begin{aligned} &v' = v + 1, \quad \Delta v' = \Delta v + 1, \\ &q' = q - \lceil [(m^{v'}_d - m^v_d) \times f]/bw' \rceil, \\ &\Delta q' = q' - q, \quad d' = d. \end{aligned} \quad (4)$$

Similarly, we can derive other state transition probabilities.

The reward in MDP is the payoff obtained when a particular action is taken at a state:

$$r_{t+1} = R(s_t = s), \quad (5)$$

where R maps the state to a reward. Table 1 lists the rewards defined for different states. $*$ means any value for the state, and F^+ represents that the number of buffered frames is larger than $B_T \times N_s$. The reward of a state can be looked up in the table from the top to the bottom, using the reward of the first entry in the table matching the current state. The values of rewards need to be carefully designed, since it is closely related to the control objective. The stored video has a finite length, and when the state reaches $d = N_T$, i.e., all the segments have been downloaded, the streaming task completes, which is called an episodic task. Therefore, we give state $(*, *, *, *, *, N_T)$ a reward of 0. Besides, any action taken in this state will not change the state, i.e., the terminal state will not affect the decision process. By giving the minimum reward when the buffer is empty, we can minimize playback interruption; by giving a negative reward to the state when the number of buffered frames is larger than the desired value, we can avoid buffer overflow. When both Δq and Δv are 0, the maximum reward (0) is given, since in these states, the playback will be smooth and the selected video version matches the available bandwidth well.

Table 1
Rewards associated with states.

$s_t = s$	$R(s)$
$(*, *, *, *, *, N_T)$	0
$(0, *, *, *, *, *)$	$-F + \Delta q$
$(F^+, *, *, *, *, *)$	$-F - \Delta q$
$(*, \Delta q, *, \Delta v, *, *)$	$\min(-\alpha\lvert\Delta v\rvert, -\lvert\Delta q\rvert)$

In addition, we can associate a weight parameter α with the reward to make a trade-off between the average playback quality and playback smoothness. When α is smaller, the video streaming can be more adaptive to the available bandwidth to achieve a higher average playback quality; when α is larger, a higher priority is given to the playback smoothness. Note that the reward is independent of the bandwidth, since we are unable to control the actual varying bandwidth.

4. Algorithm design

4.1. Optimal solution

We formulate the rate adaptation problem as an optimization problem. The objective is to find a strategy $\pi(s)$ for the action taken at a state s to maximize the reward received in the long run. Given a deterministic strategy, the state-value function is thus

$$V^\pi(s) = \sum_{s'} \mathcal{P}^a_{ss'}[R(s) + \gamma V^\pi(s')], \quad (6)$$

where γ is the discounting rate $0 \le \gamma \le 1$. Note that in our case, we can set γ to 1, since we are dealing with an episodic streaming task. An optimal strategy $\pi^*(s)$ should maximize the state-value function in the long run, i.e.,

$$\pi^*(s) = \arg\max_\pi \sum_{s'} \mathcal{P}^a_{ss'}\left[R(s) + \gamma V^*(s')\right], \quad (7)$$

where $V^*(s)$ is the optimal value function. Then, we can obtain the optimal streaming strategy using a value iteration algorithm [6]. The solution is a table that maps each state to an optimal action. Furthermore, to reduce the number of states for MDP and the input size for value iteration, we divide the buffer size (in frames) into a number of bins and index them as q_b starting from 0 to $\lfloor B_T \times N_s/BS \rfloor$, where BS is the number of frames in each bin. Then we use $q_b \times BS$ to represent the number of buffered frames for each bin.

4.2. Online real-time algorithm

The optimal streaming algorithm can provide us important insights for dynamic rate adaptation, but it has several limitations which make it less practical. First, the optimal streaming algorithm requires the knowledge of the available bandwidth statistics, i.e., the bandwidth state and transition probability between states. This information is difficult to obtain or estimate beforehand. Using finite state Markov models for dedicated wireless channels is one way to obtain the channel statistics, but the available bandwidth statistics for shared wireless access links or backbone links should depend on not only the physical channel dynamics but also the less-predictable contention from other users. Second, we rely on the value iteration algorithm to solve (7), and the complexity of these algorithms are proportional to the number of states. For the optimal video rate adaptation problem, the number of states can be so large that the computational complexity makes it difficult, if not impossible, to be used in real time.

One category of online algorithms, such as Q-Learning and Saras [6], does not require the complete knowledge of the system dynamics but need to repeat the streaming process many times, and take a long time to improve the policy. Thus, they are not practical for our problem as well.

In the following, we propose an online algorithm integrating the learning and planning process, which learns bandwidth statistics and makes decisions in real time, and the reasoning of the proposed algorithm still comes from reinforcement learning. The rate adaptation algorithm is decomposed into two modules, bandwidth statistics estimation and real-time action search. After one segment is received, the bandwidth statistics estimation module will update the average bandwidth and transition probability, and then the real-time search module will determine the best action based on the bandwidth statistics and the current state. We will describe these two modules in the following subsections.

4.2.1. Bandwidth statistics estimation

In order to obtain the bandwidth statistics, we divide the bandwidth into $L+1$ regions (states) based on the average bitrate of video layers. $\bar{r}_i$ is the average bitrate of layer i. The regions are $[0,\bar{r}_1]$, $(\bar{r}_1,\bar{r}_2]$, $(\bar{r}_2,\bar{r}_3]$, …, and $(\bar{r}_L,\infty]$. When the d-th segment is received completely, we can calculate the effective throughput l_d for downloading this segment and determine which bandwidth region (state) it falls in. We define a transition count matrix C whose element c_{ij} denotes the number of bandwidth transitions from state i to j. We can calculate the transition probability as

$$P_{ij}=\frac{c_{ij}+k}{\sum_{k=1}^{L+1}c_{ik}+k(L+1)}, \tag{8}$$

where k is the Laplacian smoothing parameter. Laplacian smoothing can avoid over-fitting and naturally initialize the transition probability as equal. The average bandwidth $\bar{b}_i$ of state i is

$$\bar{b}_i=\frac{\sum_d I(l_d,i)}{\sum_{k=1}^{L+1}c_{ki}}, \tag{9}$$

where $I(l_d,i)$ is defined as

$$I(l_d,i)=\begin{cases} l_d & \text{if } l_d\in(\bar{r}_{i-1},\bar{r}_i],\\ 0 & \text{otherwise,}\end{cases} \tag{10}$$

In this way, as the number of received segments increases, the bandwidth statistics will be more accurate.

4.2.2. Real-time action search

The proposed real-time (RT) search algorithm is listed in Algorithm 1, where D is the search depth threshold to define how many steps we look forward into the future:

$$\pi^*(s)=\arg\max_a Q^*(s,a), \tag{11}$$

$$V^*(s)=\max_a Q^*(s,a), \tag{12}$$

$$Q^*(s,a)=\sum_{s'}\mathcal{P}^a_{ss'}\left[R(s)+\gamma V^*(s')\right]=R(s)+\gamma\sum_{s'}\mathcal{P}^a_{ss'}V^*(s'). \tag{13}$$

The reasoning behind the real-time search algorithm is from (11)–(13). We can see the recursive relationship between the optimal state-value function $V^*(s)$ and the optimal action-value function $Q^*(s,a)$, where $Q^*(s,a)$ is the long-term return when the state is s and action a is taken. Eqs. (11) and (12) are equivalent since they both indicate that action a which maximizes $Q^*(s,a)$ is the optimal action for state s. While in (13), $Q^*(s,a)$ is dependent on the expected optimal value function of the next state s'. In the online algorithm, procedure OptStateValue corresponds to (12) and procedure OptActionValue corresponds to (13). In line 13, the procedure OptStateValue returns the best action for state s and the long-term return. In line 17, the state s' can be obtained from s and a based on the transition models defined in (3) and (4).

Algorithm 1. Real-time search algorithm.

```
1:  procedure OptStateValue(s, k)
2:    if k ≥ D then
3:      return R(s)
4:    end if
5:    max ← −∞
6:    for all a ∈ A(s) do
7:      q ← OptActionValue(s, a, k)
8:      if q > max then
9:        best ← a
10:       max ← q
11:     end if
12:   end for
13:   return (best, max)
14: end procedure

15: procedure OptActionValue(s, a, k)
16:   q ← R(s)
17:   for all s' from s, a do
18:     (best, v) ← OptStateValue(s', k+1)
19:     q ← q + γ P_ss' v
20:   end for
21:   return q
22: end procedure
```

Essentially, the states can be viewed as the nodes of a search tree and the possible combinations of actions and bandwidth transitions determine the edges of the tree. The online algorithm is traversing the tree, and the time

complexity of the online algorithm is $O(b^D)$, where b is the number of branches of the tree (the product of the number of actions and possible bandwidth transitions). If the search depth is equal to the whole length of a video, then we can obtain the optimal solution. But the computation is too high to obtain such a result in real-time, so the search depth is set to a small value in the proposed real-time search algorithm. Thus, the algorithm gives a suboptimal solution. We use D to make a trade-off between the optimality and computational complexity. In Section 5, we demonstrate that the performance of the proposed real-time algorithm is close to the optimal one when D is 3, and the computation cost is low enough for real-time decision-making. As for the memory consumption, since the real-time search algorithm is similar to the depth-first search, the memory consumption is bounded by $O(D)$, i.e., the search depth. Therefore, when D is small, the memory consumption is small too.

5. Performance evaluation

In this section, we first define the objective QoS metrics in terms of playback interruption, average playback quality and playback smoothness. Then we describe the evaluation framework including the layered video storage structure, SVC video player implementation details and the experiment settings. We evaluate the proposed online algorithm and compare it with the optimal solution and the existing state-of-the-art rate adaptation algorithm [9] by experiments and simulations.

5.1. QoS metrics

There are many different approaches and metrics for video streaming quality assessment, ranging from referenced to non-referenced ones, and subjective to objective. Here we focus on the quality of experience perceived by end users, i.e., how often the streaming is interrupted, the average video quality and how often the quality changes, in an objective manner. Such QoS metrics highly rely on the rate adaptation algorithm used and can lead to subjective metrics if needed.

(1) *Interruption ratio*: Every $1/f$ second (f is the video frame rate), the video player displays one frame, which is defined as one display event. If there is no decoded frame available to display, a playback interruption occurs. Let n_0 be the number of occurrences that a frame to be displayed is not available. Denote by n_t the total number of display events, the interruption ratio (IR) is defined as $\mathrm{IR} = n_0/n_t$. Among the performance metrics, we give the IR the highest priority because interruptions during playback are most unpleasant for users. It means that if the IR of an algorithm is higher than the other, then this algorithm is inferior no matter how good the other performance metrics are.

(2) *Average playback quality*: We define a continuous playback of layer i video as one run and its length in terms of the number of display events as n_r for the r-th run. There are totally N runs. The layer index 0 denotes that a playback interruption occurs. The weighted sum of the layer index is used to measure the average playback quality (APQ), which is defined as $\mathrm{APQ} = \sum_{r=1}^{N}(n_r \times i)/\sum_{r=1}^{N}(n_r)$. We also use the PSNR metric to measure the average playback quality. The experiment results show that the APQ has a highly positive correlation with PSNR.

(3) *Playback smoothness* [22]: Intuitively, a longer run length leads to a smoother watching experience. The mean square root of run length is used to measure the playback smoothness (PS), and we have $\mathrm{PS} = \sqrt{\sum_{r=1}^{N}(n_r)^2/N}$. It also gives a fair evaluation when the length of one run is much larger than the others, when compared with the arithmetic average.

5.2. Evaluation framework and testbed settings

In this section, we describe the layered video storage structure at the server side, video player implementation details, and experiment and testbed settings.

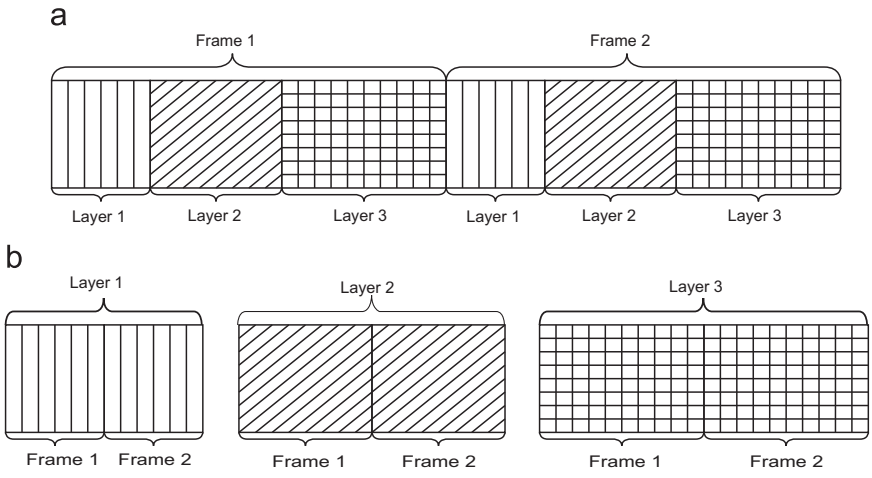

Fig. 2. Layered video storage structure.

5.2.1. Layered video storage structure

At the server side, the video is segmented and encoded into multiple layers. To minimize the server load, the server stores the SVC video in a layered segment structure. Fig. 2(a) shows the structure of the original bitstream generated by an SVC encoder (supposing that each segment has two frames and there are three layers for illustration purposes). In the server, the Network Abstraction Layer (NAL) units of all the frames with the same layer index are stored together as shown in Fig. 2(b). When the client receives the layer segments, it can reorganize them into the original bitstreams for decoding and playback.

This layered segment storage structure can better utilize the web caching infrastructure and the client can request any layer segment flexibly. Since the frames of different layers are separately stored, the client uses HTTP pipelining to request several layer segments and construct the video. Other solutions including partial HTTP requests or letting the web server extract the requested layer segments on-the-fly not only add the complexity to the server, but also slow down the response time to the client requests. One concern with the proposed storage structure is that the client needs to wait until all the layers in a frame are downloaded before playback. A solution to minimize the latency is that the client can establish parallel TCP connections to request the different layers simultaneously, which is left for future investigation. Due to the limited number of frames in a segment and usually at least one segment of frames retrieved before playback starts, such a tradeoff is considered acceptable as the experiment shows.

5.2.2. Video player implementation

The client-side video player is implemented using an open-source SVC decoder [23]. As depicted in Fig. 3, the video player consists of three modules, rate adaptation, decoder and display. The rate adaptation module determines the version of the next video segment to request. The decoder fetches a segment from the segment buffer and decodes the segment as fast as possible and stores the decoded picture in the picture buffer. The decoder will not fetch a new segment until the number of pictures left to be displayed is smaller than a threshold. In this way, the buffered segments have a chance to be enhanced with higher-layer segments that may arrive later. The display module simply fetches a picture from the buffer and displays it on the screen every $1/f$ seconds. Meanwhile, the video player also collects the system state information including the buffer state and the playback version index.

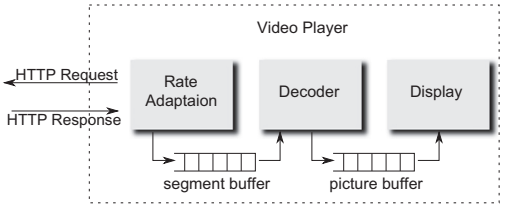

Fig. 3. Video player structure.

Table 2
Layer configuration.

Resolution	Avg. bit-rate (Kbps)	std bit-rate deviation	Y-PSNR (upscaled)	Layer index
320 × 180	112.84	39.01	35.47 (30.99)	1
320 × 180	238.94	88.84	39.44 (32.62)	2
640 × 360	363.82	140.33	35.90 (35.90)	3

5.2.3. Experiment and testbed settings

We used the open-source SVC codec JSVM [24] to encode the sample video ("Big Buck Bunny" [25]) into three layers, and their configurations are listed in Table 2. The encoding rate of layer n is the cumulative rate of all the layers up to n. Note that the Y-PSNR of Layer 3 is lower than that of Layer 2, but we still prefer Layer 3 video which has a higher resolution, as it leads to a better watching experience when displayed on a larger screen due to a higher dots per inch (DPI) index. Typically, PSNR reflects the mean square errors between the original video signal and the received signal; if we use the original videos with different layers, the layer-wise PSNR comparison becomes unfair as a received lower-layer video with a higher PSNR may have a worse visual quality. To make the PSNR comparison meaningful with layered video, we upscale the lower resolution 320 × 180 video to 640 × 360 video using the "bicubic" interpolation method and then calculated the PSNR accordingly. The PSNRs for the 3 layers from low to high quality are 30.99, 32.62 and 35.90 dB. In this way, the scaled PSNR can reflect the visual quality of layered videos consistently from a user's experience point of view.

Each layer is chopped into small segments of 17 frames. The total number of segments, N_T, is 200, and the frame rate is 24 frames per second. From the experiments, we find that the segment size of 17 frames is small enough to react to the varying bandwidth, and large enough to keep the HTTP overhead low. The target buffer size is $B_T = 20$ segments. The playback starts when 4 segments are received.

Fig. 4 shows the testbed configuration. The testbed used `Lighttpd` as the streaming web server. The server and the wireless router were connected via wired links, and two laptops accessed the web server through the wireless router. The laptop C1's CPU is dual-core at 2.53 GHz with 2 GB memory, and for another laptop C2, the CPU is dual-core at 2.26 GHz with 4 GB memory. OpenWRT has been installed on the wireless router, configured in the IEEE 802.11b mode, and the downlink transmission rate was set to 1 or 2 Mbps to emulate a dynamic wireless environment with different congestion levels.

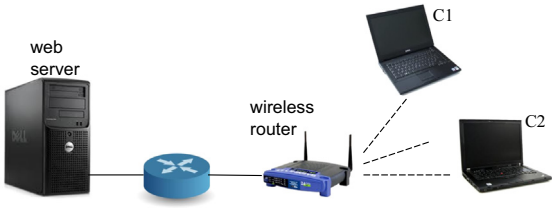

Fig. 4. Network topology for the experiments.

5.3. Experiments and simulations

5.3.1. Experiment 1 (no background traffic)

In this case, the transmission rate of the wireless router was set to 1 Mbps, and the effective goodput was about 717.6 Kbps (measured by downloading a large file through HTTP). We conducted the experiment for 10 runs and the presented performance results are the average values.

In Table 3, we compare the proposed online realtime algorithm, RT, with the rate adaptation algorithm, RA, proposed in [9] and the fixed layer algorithm, FL(3), which always requests all three layers. Generally, when the search depth D is larger, the performance of RT is better. But we cannot increase it too much. For C1, when $D=1$ or 2, it takes less than 1 ms to make the decision for a new segment. When $D=3$, it takes about 10 ms, still less than the duration of one frame. When $D=4$, it takes 240 ms to make one decision, which is approximately the playback time of six frames. We believe that the decision time less than the playback time of one frame is acceptable. Therefore, in the following experiments, D is set to 3 by default. (For C2, we also set D to be 3, although its CPU cycle is slightly lower than C1).

For the proposed RT algorithm with $D=3$, as we increase the weight parameter α to 12 to emphasize on higher smoothness, when compared with $\alpha=10$, we can see that APQ is reduced but PS is increased, which shows that α can make a trade-off between APQ and PS.

Comparing RT ($D=3$) and RA, we note that RT can achieve both a higher APQ and PS, while maintaining a lower queue length. Since the available bandwidth is sufficient to support Layer 3 video, FL(3) outperforms the RT and RA algorithm. However, the performance of FL will degrade dramatically when there is any competing traffic and the available bandwidth is more dynamic. We can see this in the following experiments.

Table 3
Experiment 1 results.

Algorithm	IR	APQ (PSNR)	PS	Max queue
RA	0	2.57 (34.73)	260.80	21.0
RT $\alpha=10$ $D=1$	0	2.24 (33.90)	700.96	19.0
RT $\alpha=10$ $D=2$	0	2.63 (34.88)	258.996	19.0
RT $\alpha=10$ $D=3$	0	2.72 (35.07)	643.60	19.0
RT $\alpha=10$ $D=4$	0	2.45 (34.36)	905.80	19.0
RT $\alpha=10$ $D=3$	0	2.63 (34.85)	1132.63	19.0
FL(3)	0	3 (35.90)	3400	21.0

Table 4
Experiment 2 results.

Algorithm	IR	APQ (PSNR)	PS	Max queue
RA	0	1.19 (31.33)	179.98	20.7
RT $\alpha=10$	0	1.29 (31.50)	529.88	18.6
FL(1)	0	1 (30.99)	3400	20.9
FL(2)	0.27	1.46 (32.62)	34.11	5.2

5.3.2. Experiment 2 (with background traffic)

In this experiment, the transmission rate of the wireless router was also set to 1 Mbps. Laptop C1 ran the SVC Player, and another laptop C2 downloaded a large file from the web server. From Table 4, we can see that the available bandwidth is sufficient to support Layer 1 video but cannot support Layer 2 video, since for FL(2) algorithm, the IR is as high as 0.27. Because we give the IR the highest priority when comparing rate adaptation algorithms, FL(2) is inferior compared to RA and RT, although its APQ value is better than the others. Both RA and RT algorithm can achieve APQ between 1 and 2, and RT is better than RA in terms of both APQ and PS.

5.3.3. Experiment 3 (competing video flows with on–off background traffic)

In this experiment, the transmission rate of the wireless router was set to 2 Mbps. Each laptop ran the SVC player and an on–off background traffic flow. The on–off traffic flow downloaded a file (1.3 MB) from the server, and then slept for 10 s, and this process repeated until the end of the experiment. Since the time needed to download the fixed-size file varies due to contention, there can be 0–2 background flows during the experiment, which makes the available bandwidth more dynamic. In order to evaluate the fairness of the rate adaptation algorithms, both laptops run the same rate adaptation algorithm. In Table 5, we can see from the FL algorithm that the video version can be supported between Layer 1 and Layer 2. RT is better than RA in terms of both APQ and PS, and the two videos on different laptops have roughly the same QoS performance, which demonstrates that the proposed RT algorithm together with the TCP congestion control can allow

Table 5
Experiment 3 results.

C	Algorithm	IR	APQ (PSNR)	PS	Max queue
C1	RA	0	1.46 (31.93)	146.33	21
	RT $\alpha=10$	0	1.56 (32.04)	269.08	19
	FL(1)	0	1 (30.99)	3400	21.0
	FL(2)	0.05	1.89 (32.62)	545.55	19.1
C2	RA	0	1.56 (32.07)	141.29	21
	RT $\alpha=10$	0	1.59 (32.11)	321.89	19
	FL(1)	0	1 (30.99)	3400	20.8
	FL(2)	0.02	1.95 (32.62)	1040.64	19.9

Table 6
Experiment 4 results.

Algorithm	IR	APQ (PSNR)	PS	Max queue
RA	0	2.07 (33.1)	120.94	21
RT	0	2.18 (33.2)	318.31	19
FL(3)	0.06	2.82 (32.75)	541.02	15.2

the competing videos to obtain a fair share of the bandwidth.

Table 7
Simulation results.

Simu	Algorithm	IR	APQ (PSNR)	PS	Max queue
S1	RA	0	2.35 (34.17)	153.48	21
	RT $\alpha=10$	0	2.57 (34.66)	517.98	19
	OS $\alpha=2$	0	2.82 (35.24)	1434.99	19.4
	FL(3)	0	3 (35.90)	3400	21.0
S2	RA	0	1.34 (31.56)	84.59	21
	RT $\alpha=10$	0	1.78 (32.32)	300.49	19
	OS $\alpha=2$	0	1.80 (32.33)	498.5	17.5
	FL(1)	0	1 (30.99)	3400	20.8
	FL(2)	0.09	1.82 (32.62)	193.13	9.1

5.3.4. Experiment 4 (performance study over long-distance connections)

In this experiment, we set up the web server on the campus network and the video player client was located outside the campus network through commercial Internet. There were 14 hops between the client and the server (identified by the traceroute program) and the last hop was wireless link. In order to make the network bandwidth more dynamic, we created 35 flows of on–off background traffic in the network. Each flow repeatedly requested a 2 MB file from the server and slept for 10 s. Table 6 shows the experiment results. The available bandwidth cannot support Layer 3 video, so FL(3) suffers from playback interruptions. Both RA and RT can achieve APQ between 2 and 3 and RT outperforms RA in terms of both APQ and PS.

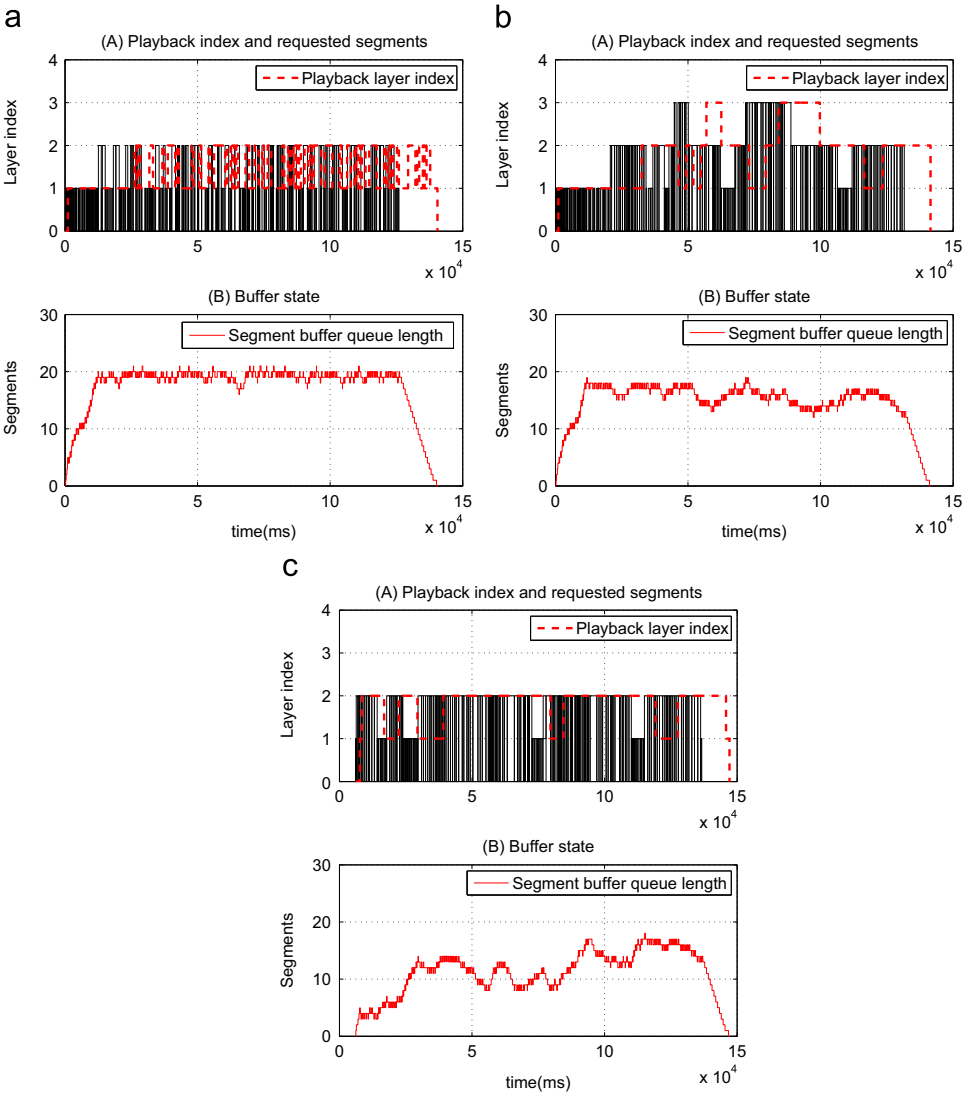

Fig. 5. Playback trace comparison: (a) RA, (b) RT, and (c) OS.

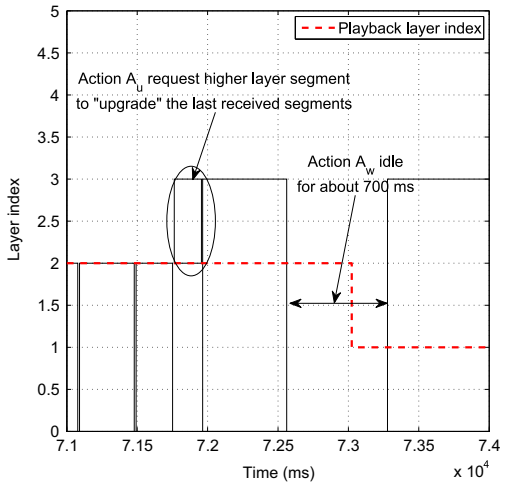

Fig. 6. A zoom-in of playback trace of RT.

5.3.5. Simulation results

We also evaluate the gap between the proposed online RT algorithm with the optimal streaming policy, OS, obtained by dynamic programming. Since the optimal streaming policy cannot be generated in real time, we used the trace-driven simulation to compare these algorithms.

The traces share the same bandwidth statistics with Experiments 1 and 2, denoted as S1 and S2, respectively. The statistics of the average bandwidth for different states and the transition probability are obtained off-line as the input to the dynamic programming. Similar to the experiment, the results are the average over 10 runs. Optimal solution's queue length is in the unit of segments, while that of online algorithm is in the number of frames. Since the queue variation unit of OS and RA is in different unit, α of these two algorithms are set differently. From Table 7, both RT and OS are better than RA, and there exists a small performance gap between RT and OS, which indicates the trade-off between the performance and the computational complexity.

Fig. 5 shows the playback trace of one run in simulation S2. We can see from the figure that RA suffers from frequent layer switching, although it keeps more segments in the buffer than RT and OS, which indicates that buffer fullness does not directly reflect video quality. The proposed online algorithm, RT, is more aggressive than the optimal OS algorithm. Although the APQ of RT and OS in this run are both around 1.78, OS achieves a higher PS by never requesting Layer 3 video, and the average queue length of OS is smaller.

We further zoom in the playback trace for RT. In Fig. 6, the rectangle represents a segment and the width of it denotes the download time duration (from the time instant of sending out the HTTP request to that of receiving the segment completely). The horizontal gap between the edges of the rectangles is due to the waiting action to avoid buffer overflow. Fig. 6 shows the advantage of the proposed video streaming framework using SVC: the rectangles that rise from a non-zero layer index (circled and annotated by arrows) are the layer segments to "upgrade" the already buffered segments to improve both APQ and PS, which is not possible when using the traditional AVC streaming techniques.

Table 8
Larger segment size.

Segment size (frames)	Algorithm	IR	APQ (PSNR)	PS	Max queue
17	RA	0	1.34 (31.56)	84.59	21
	RT	0	1.78 (32.32)	300.49	19
51	RA	0	1.32 (31.35)	184.63	22
	RT	0	1.65 (32)	372.26	19.1
102	RA	0	1.31 (31.52)	319.27	23
	RT	0	1.47 (31.8)	389.75	22.2

Table 9
Experiment 5 results.

T	Algorithm	IR	APQ (SSIM)	PS	Max queue	Wasted seg
T1	RT	0	1.92 (0.96)	1959.86	17.5	0
	BIEB	0	1.39 (0.93)	509.39	13	1.0
	FL(2)	0.66	1.03 (1.0)	545.55	3.1	0
T2	RT	0	2.75 (0.98)	3398.24	17.4	0
	BIEB	0	2.70 (0.98)	2149.69	20	1.1
	FL(3)	0.04	2.86 (1.0)	2486.02	20	0

In order to evaluate the impact of segment size on the proposed algorithm, we compare the algorithms using a larger segment size with the same settings as those in S2. The results are shown in Table 8. From the table, with a larger segment size, the playback is smoother, since the minimum length with a constant playback quality is increased. On the other hand, a larger segment size also slows down the response to varying bandwidth; therefore, the APQ for both algorithms is reduced. Nevertheless, RT still shows the advantage over RA in terms of both APQ and PS. As we can imagine, when the segment size reaches the size of the whole video, there is no chance of performing rate adaptation and all these algorithms fall back to the FL algorithm.

5.3.6. Experiment 5 (under simulated varying wireless bandwidth)

To further demonstrate the performance of the proposed algorithm, we compared with another state-of-the-art work [26]. In this simulation, we used bandwidth traces and HTTP throttling module in Apache from [27] to simulate the varying wireless bandwidth. We have chosen two bandwidth traces to represent wide range of network conditions. The average bandwidth of the first trace T1 is about 1829 Kbps and the average bandwidth of the second trace T2 is about 3774 Kbps. We used the same SVC video trace "Tears of Steel" as in [26]. We also adopted the same quality metric Structural Similarity (SSIM) index for fair comparison. We also introduced one more performance

metric, wasted segments, from [26]. It denotes the number of segments downloaded but not played.

The experiment results are summarized in Table 9. From the results, we can see that for fixed-layer algorithms, there are playback interruptions. This is the reason why we need adaptive algorithms. For RT and BIEB that proposed in [26], we can see that in both traces, RT outperforms BIEB in terms of both the average playback quality and smoothness. In addition, all the segments downloaded are played using RT algorithm.

6. Conclusions

In this paper, for DASH-based adaptive video streaming in wireless networks, we have formulated the rate adaptation problem as an MDP problem and used dynamic programming to obtain the optimal streaming policy. To reduce the complexity of the optimal streaming policy, we have proposed an online algorithm which learns the bandwidth statistics and makes the request decisions for the near future. The trade-off between the average video quality and playback smoothness can be made by adjusting the parameter in the reward function. Experiment results have shown that the proposed solution is feasible and can substantially outperform the existing one. There are several issues worth further investigation. For instance, to fully utilize the layered feature of SVC, we may consider other possible actions, such as to "upgrade" multiple previously received segments when possible. We may use a moving window to better estimate the bandwidth state transition probability and capture the non-stationary behavior of varying bandwidth. We may also keep a history of the layer index for requested segments and make decisions based on these records to further improve the smoothness of video playback and the user-perceived quality of experience.

References

[1] S. Xiang, L. Cai, J. Pan, Adaptive scalable video streaming in wireless networks, in: ACM MMSys'12, Chapel Hill, NC, USA, 2012, pp. 167–172.
[2] Cisco, Cisco Visual Networking Index: Forecast and Methodology, 2013–2018.
[3] L. Plissonneau, E. Biersack, A longitudinal view of http video streaming performance, in: ACM MMSys'12, Chapel Hill, NC, USA, 2012, pp. 203–214.
[4] T. Stockhammer, Dynamic adaptive streaming over HTTP: standards and design principles, in: ACM MMSys'11, San Jose, CA, USA, 2011, pp. 133–144.
[5] Y. Sánchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, Y. Lelouedec, iDASH: improved dynamic adaptive streaming over HTTP using scalable video coding, in: ACM MMSys'11, San Jose, CA, USA, 2011, pp. 257–264.
[6] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, The MIT Press, Cambridge, Massachusetts 1998.
[7] M. Kobayashi, H. Nakayama, N. Ansari, N. Kato, Robust and efficient stream delivery for application layer multicasting in heterogeneous networks, IEEE Trans. Multimed. 11 (1) (2009) 166–176.
[8] A. Begen, T. Akgul, M. Baugher, Watching video over the web: Part 1: Streaming protocols, IEEE Internet Comput. 15 (2) (2011) 54–63.
[9] C. Liu, I. Bouazizi, M. Gabbouj, Rate adaptation for adaptive HTTP streaming, in: ACM MMSys'11, San Jose, CA, USA, 2011, pp. 169–174.
[10] S. Akhshabi, A.C. Begen, C. Dovrolis, An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP, in: ACM MMSys'11, San Jose, CA, USA, 2011, pp. 157–168.
[11] L. De Cicco, S. Mascolo, V. Palmisano, Feedback control for adaptive live video streaming, in: ACM MMSys'11, San Jose, CA, USA, 2011, pp. 145–156.
[12] A. Bokani, M. Hassan, S. Kanhere, HTTP-based adaptive streaming for mobile clients using markov decision process, in: IEEE PV'13, 2013, pp. 1–8.
[13] Z. Yan, C.W. Chen, B. Liu, Admission control for wireless adaptive HTTP streaming: An evidence theory based approach, in: ACM MM'14, 2014, pp. 893–896.
[14] L. Cai, S. Xiang, Y. Luo, J. Pan, Scalable modulation for video transmission in wireless networks, IEEE Trans. Veh. Technol. 60 (9) (2011) 4314–4323.
[15] X. Zhu, R. Pan, M. Prabhu, N. Dukkipati, et al., Layered internet video adaptation (liva): network-assisted bandwidth sharing and transient loss protection for video streaming, IEEE Trans. Multimed. 13 (4) (2011) 720–732.
[16] T. Schierl, Y. Sanchez de la Fuente, R. Globisch, C. Hellge, T. Wiegand, Priority-based media delivery using SVC with RTP and HTTP streaming, Multimed. Tools Appl. 55 (2011) 227–246.
[17] M. Xing, S. Xiang, L. Cai, A real-time adaptive algorithm for video streaming over multiple wireless access networks, IEEE J. Sel. Areas Commun. 32 (4) (2014) 795–805.
[18] X. Wang, J. Chen, A. Dutta, M. Chiang, Adaptive video streaming over whitespace: SVC for 3-tiered spectrum sharing, in: IEEE Infocom'15, 2015.
[19] B. Cheng, R.J. Stanley, S. Antani, G.R. Thoma, A novel computational intelligence-based approach for medical image artifacts detection, in: SDIWC AIPR'10, 2010, pp. 113–20.
[20] H.S. Wang, N. Moayeri, Finite-state Markov channel—a useful model for radio communication channels, IEEE Trans. Veh. Technol. 44 (1) (1995) 163–171.
[21] R. Zhang, L. Cai, A packet-level model for UWB channel with people shadowing process based on angular spectrum analysis, IEEE Trans. Wirel. Commun. 8 (8) (2009) 4048–4055.
[22] S. Nelakuditi, R. Harinath, E. Kusmierek, et al., Providing smoother quality layered video stream, in: ACM NOSSDAV'00, 2000.
[23] M. Blestel, M. Raulet, Open SVC decoder: a flexible SVC library, ACM MM'10, New York, NY, USA, 2010, pp. 1463–1466.
[24] J. Reichel, H. Schwarz, M. Wien, Joint scalable video model 11 (JSVM 11), Joint Video Team, Doc. JVT- X.
[25] Big Buck Bunny, ⟨http://www.bigbuckbunny.org⟩.
[26] C. Sieber, T. Hosfeld, T. Zinner, P. Tran-Gia, C. Timmerer, Implementation and user-centric comparison of a novel adaptation logic for DASH with SVC, in: IFIP/IEEE IM'13, 2013, pp. 1318–1323.
[27] HSDPA-Bandwidth Logs for Mobile HTTP Streaming Scenarios, ⟨http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/⟩.