A Real-Time Adaptive Algorithm for Video Streaming over Multiple Wireless Access Networks

Min Xing, *Student Member, IEEE,* Siyuan Xiang, *Member, IEEE,* and Lin Cai, *Senior Member, IEEE*

***Abstract*—Video streaming is gaining popularity among mobile users. The latest mobile devices, such as smart phones and tablets, are equipped with multiple wireless network interfaces. How to efficiently and cost-effectively utilize multiple links to improve video streaming quality needs investigation. In order to maintain high video streaming quality while reducing the wireless service cost, in this paper, the optimal video streaming process with multiple links is formulated as a Markov Decision Process (MDP). The reward function is designed to consider the quality of service (QoS) requirements for video traffic, such as the startup latency, playback fluency, average playback quality, playback smoothness and wireless service cost. To solve the MDP in real time, we propose an adaptive, best-action search algorithm to obtain a sub-optimal solution. To evaluate the performance of the proposed adaptation algorithm, we implemented a testbed using the Android mobile phone and the Scalable Video Coding (SVC) codec. Experiment results demonstrate the feasibility and effectiveness of the proposed adaptation algorithm for mobile video streaming applications, which outperforms the existing state-of-the-art adaptation algorithms.**

***Index Terms*—DASH, Markov decision process, video streaming, multiple links.**

I. Introduction

VIDEO streaming is gaining popularity among mobile users recently. Considering that the mobile devices have limited computational capacity and energy supply, and the wireless channels are highly dynamic, it is very challenging to provide high quality video streaming services for mobile users consistently. It is a promising trend to use multiple wireless network interfaces with different wireless communication techniques for mobile devices. For example, smart phones and tablets are usually equipped with cellular, WiFi and Bluetooth interfaces. Utilizing multiple links simultaneously can improve video streaming in several aspects: the aggregated higher bandwidth can support video of higher bit rate; when one wireless link suffers poor link quality or congestion, the others can compensate for it.

High resilience to bandwidth variation and easy deployment are both important requirements for video streaming applications. Currently, progressive download, one of the most popular and widely deployed streaming techniques, buffers a large amount of video data to absorb the variations of bandwidth. Meanwhile, as video data are transmitted over HTTP protocols, the video streaming service can be deployed on any web server. However, the video quality version can only be manually selected by users and such decision can be error-prone. Since the smart phones only have limited storage space, it is impractical to maintain a very large buffer size. In addition, the buffered unwatched video may be wasted if the user turns off the video player or switches to other videos. Furthermore, progressive download typically does not support transmitting video data over multiple links.

To overcome the above disadvantages of progressive download, dynamic adaptive streaming over HTTP (DASH) [1] has been proposed. In a DASH system, multiple copies of pre-compressed videos with different resolution and quality are stored in segments. The rate adaptation decision is made at the client side. For each segment, the client can request the appropriate quality version based on its screen resolution, current available bandwidth, and buffer occupancy status. This pull-based DASH scheme can be extended to support multiple links, *i.e.*, we can let the client request different parts of one segment over different links. How to optimize this rate adaptation process for video streaming over multiple wireless links, considering the video quality of service (QoS) requirements, the wireless channel profiles, and the wireless service costs of multiple links is an open issue.

In this paper, we formulate the multi-link video streaming process as a reinforcement learning task. For each streaming step, we define a state to describe the current situation, including the index of the requested segment, the current available bandwidth and other system parameters. A finite-state Markov Decision Process (MDP) can be modeled for this reinforcement learning task. The reward function is carefully designed to consider the video QoS requirements, such as the interruption rate, average playback quality, and playback smoothness, as well as the service costs. To make a trade-off between different QoS metrics and the cost, we can adjust the parameters of the reward function. To solve the MDP in real time, we proposed an adaptive best-action search algorithm to obtain a sub-optimal solution. A realistic testbed is implemented to better evaluate the performance of our solution.

The main contributions of this paper are threefold. First, we formulate the video streaming process over multiple links as an MDP problem. To achieve smooth and high quality video streaming, we define several actions and reward functions for each state. Second, we propose a depth-first real-time search algorithm. The proposed adaptation algorithm will take several future steps into consideration to avoid playback interruption and achieve better smoothness and quality. Last, we implement a realistic testbed using an Android phone and Scalable Video Coding (SVC) encoded videos to evaluate the performance.

Manuscript received April 18, 2013; revised September 22, 2013. Preliminary results of this work have been presented at IEEE Globecom'12 [26].

M. Xing, S. Xiang and L. Cai are with the Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada (e-mail: {mxing,siyxiang,cai}@ece.uvic.ca).

Digital Object Identifier 10.1109/JSAC.2014.140411.

The experiment results show that the proposed adaptation algorithm is feasible for video streaming over multiple wireless access networks, and it outperforms the existing state-of-the-art algorithms.

The rest of the paper is organized as follows. In Section II, the related work is summarized. Section III describes the system model and the problem formulation. We present the real-time adaptive streaming algorithm in Section IV. The performance evaluation by experiments is presented in Section V, followed by the concluding remarks and further research issues in Section VI.

II. Background and Related Work

DASH has been a hot topic in recent years. There are many commercial products which have implemented DASH in different ways, such as Apple HTTP Live Streaming and Microsoft Smooth Streaming. Since the clients may have different available bandwidth and display size, each video will be encoded several times with different quality, bit rate and resolution. All the encoded videos will be chopped into small segments and stored on the server, which can be a typical web server. These small segments will be downloaded to the browsers' cache and played by the client (browser or browser plug-in). The video rate adaptation is performed at the client side, which is also called the pull-based approach. The client will determine the quality version of the requested video segment according to its current available bandwidth, resolution and the number of buffered unwatched segments. After the current segment is completely downloaded, the rate adaptation algorithm will be invoked again for the next segment.

There is extensive work covering this topic [2]–[6]. The authors in [2] proposed to estimate the bandwidth by a statistical method, and they took both the quality contribution and decoding time of each segment into consideration. K.P. Mok *et al.* presented a QoE aware DASH system [3]. Their algorithm estimates the available bandwidth by probing with the video data. In order to keep the quality level as smooth as possible, their algorithm will switch the video quality version gradually and will try to maintain the buffer level being stable. S. Akhshabi designed an evaluation method in [7] to test the performance of several existing commercial DASH products, such as Smooth Streaming, Netflix, and OSMF. In [6], T. Kupka proposed to evaluate the performance of live DASH under on/off traffic and tested four different methods to improve the performance. In [5], how to reduce unnecessary video quality variations using a probing method to identify the effective available bandwidth was given. In [4], [8], the authors designed the optimal rate adaptation algorithm for streaming scalable video coding (SVC) over HTTP using MDP. With SVC, each video frame is encoded into a base layer and several enhancement layers. Higher video quality can be achieved when more layers are received. These works only considered the single-link case, and in this work, we consider the more challenging case with multiple access links.

How to efficiently deliver video in heterogeneous wireless networks has been extensively studied, and approaches ranging from the physical layer to the application layer have been proposed [9]–[13]. Recently, a few approaches have appeared to extend the DASH technique to support multiple links. In [14], the authors summarized three typical schemes of utilizing multiple links. They compared the performance of these schemes through extensive simulations. Kaspar *et al.* proposed an approach to implementing DASH over multiple links [15]. In their algorithm, each segment will be transmitted over one link. Thus multiple segments can be transmitted at the same time. To reduce the overhead, they used HTTP pipelining to improve the performance. This approach may lead to the "last-segment problem" due to the link transmission speed difference. To overcome this disadvantage, in [16]–[18], Evensen *et al.* suggested to divide each segment into small sub-segments, and these sub-segments can be downloaded through different links. Their algorithm estimates the available bandwidth according to the throughput of the previous segment, and selects the video quality version most close to the estimated bandwidth. In the evaluation section, we will compare the performance of our proposed solution with the above state-of-the-art one [18].

III. System Model and Streaming Process Formulation

A. System Model

We consider how to utilize multiple wireless access networks together for video streaming, *e.g.*, using a combination of cellular, WiFi, and/or Bluetooth simultaneously. Here, as an example, Bluetooth and WiFi access networks are considered as we do not have end-to-end control over cellular links, and our work can be extended when other types of wireless access networks or more than two wireless access networks are used[1]. Since a wireless channel may suffer from time-varying fading, shadowing, interference and congestion, the available bandwidth of a wireless link may vary all the time. In addition, different smart phones or tablets may have different screen size and resolution. Taking these two aspects into consideration, the server should store several copies of video with different quality. The videos are encoded with SVC into a base layer and several enhancement layers, and chopped into segments and each segment can be played with a fixed duration.

We design a pull-based algorithm for video streaming, as shown in Fig. 1. After initialization, the client will request the video information which includes video resolutions, bitrates and qualities from the server through both the WiFi and Bluetooth links. The rate adaptation agent will request a video segment of appropriate quality version based on the current queue length and estimated available bandwidth. Once the request decision is made, HTTP requests over both WiFi and Bluetooth will be issued to download the video segment. This process will continue until the completion of downloading the last segment or the termination of the video streaming by the user.

[1]A promising multi-link scenario is to use both WiFi and 3G/LTE celluar links for video streaming. In our testbed, as the Android OS restricted the utilization of WiFi and 3G to access Internet simultaneously, we use WiFi and Bluetooth to test our multi-link streaming solution.

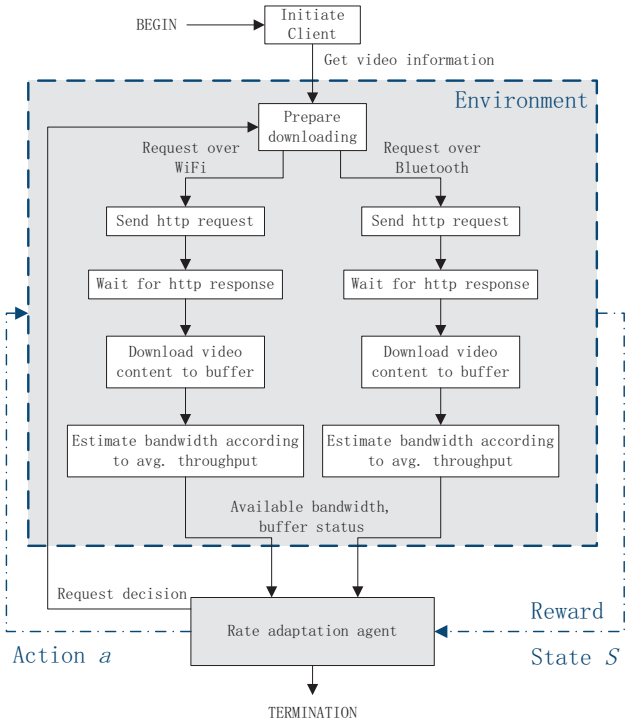


Fig. 1. System Model.

B. Streaming Process Formulation

The video streaming process can also be considered as the interaction between two modules. As shown in Fig. 1, the downloading and estimation steps in the top grey rectangle can be viewed as an integrated environment module, and the rate adaptation agent can be viewed as an agent module. The video streaming process can be formulated as a reinforcement learning task [19]. The environment sends a state signal for each video segment to the agent, and the agent will determine the best action correspondingly. For each action, the environment replies a reward to the agent. Considering the Markov property of the system states, a Markov Decision Process (MDP) can be formulated for the streaming process, and the state transition model of the Markov process needs to be devised.

We define step n as to download segment n, so the total number of steps equals the number of segments. For each step n, we define the state as $s_n = \{q_n, \Delta q_n, v_n, \Delta v_n, t_n, \Delta t_n, bw_n, bt_n, d_n\}$, with the parameters defined as follows. q_n represents the number of unplayed queued segments, which is also called the buffer level (or queue length), with the range between 0 and q_L (which is the maximum queue length). v_n is the SVC video layer index of the n-th segment. In our work, there are L SVC video layers in total, and a larger number represents a higher-quality layer. Δq_n and Δv_n are the variations of q_n and v_n respectively, *i.e.*, $\Delta q_n = q_n - q_{n-1}$, and $\Delta v_n = v_n - v_{n-1}$. The total traffic of Bluetooth used to download the previous segment is recorded by t_n, and $\Delta t_n = t_n - t_{n-1}$. The current bandwidth states of the WiFi link and the Bluetooth link are described by bw_n and bt_n, respectively. d_n indicates the current requested segment index. As the total number of video segments is N_T, d_n is in the range of $[0, N_T]$.

When the rate adaptation agent receives input of state s_n from the environment, it will make the decision of which action should the environment take. We define four types of actions, A_b, A_u, A_w, and A_s. The first two actions are for downloading the next segment at the quality determined by v, and Δv determines whether to upgrade, downgrade or maintain the current quality. Once the quality v is determined, the base layer and all enhancement layers belong to quality v will be combined together to be downloaded. Considering that drastic video quality variation will significantly decrease the perceived video quality, we avoid those drastic layer change actions by restricting the video layer change level Δv to one. The difference of A_b and A_u is that, with A_b, both the WiFi and Bluetooth links are used to download the segment simultaneously, while with A_u, only the WiFi link is used. As a short distance wireless communication technology, Bluetooth can only help the client connect to the web server indirectly via tethering cellular data services. Since cellular services are charged at much higher rates than WiFi services, it is better to limit the usage of the Bluetooth link to reduce the cost. In addition, Bluetooth connection is not quite reliable as it is easy to be interfered. Therefore, sometimes we prefer to use WiFi only. For A_b actions, the download load of WiFi and Bluetooth will be determined based on their current available bandwidth. In other words, assuming that the available bandwidth of WiFi and Bluetooth are bw and bt, the segment size is SZ, then the downloading load of WiFi is $SZ \cdot bw/(bw + bt)$. A_w represents the waiting action, and the client will wait for W seconds, equal to the duration of one segment. Since SVC video is encoded into different layers and chopped into segments to be stored on the web server, as long as the segment has not been played yet, the higher enhancement layers can be requested to improve the perceived streaming quality and smoothness. Therefore, the smooth action A_s is introduced in our work.

There are three principles to follow for the smooth action. First, when the queue length is low, the smooth action will not be taken as there is a high probability to experience playback freeze soon. Thus the smooth action will be invoked only when the queue length is sufficiently large, such as when the queue length q is larger than a certain threshold T_s. Second, for the segment which will be played soon, we will not take the smooth action, because the requested enhancement layers may miss the playback deadline. Therefore we only take the smooth action for a few number of buffered segments which are stored in the tail part of the buffer. A smooth window with size L_s is defined in our work to determine how many buffered segments will be the candidate segments for the smooth action. In our approach, the last L_s buffered segments are in the smooth window as they are least likely to miss the playback deadline. The last principle of the smooth action is to ensure that it will not bring in any additional layer variation. As shown in Fig. 2, generally, we will take the smooth action in four different scenarios. Assume the smooth window size $L_s = 6$, and thus all segments in Fig. 2 are in the smooth window. If only one segment in the smooth window has the lowest number of the enhancement layer, then we will smooth that segment by requesting the same number of enhancement layers as the other segments, which is shown in Fig. 2-(a). When there are

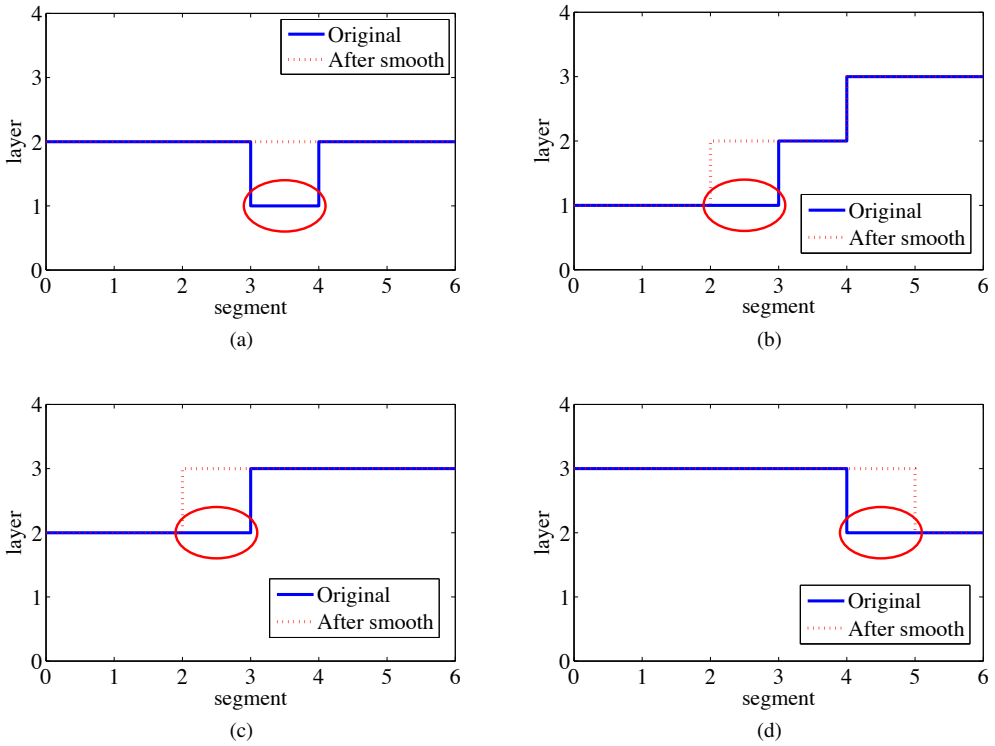


Fig. 2. Different Smooth Scenarios.

multiple layer variations in the smooth window, such as those in Fig. 2-(b), the smooth action will be applied to the segment with the smallest number of layers. To avoid additional layer variation caused by the smooth action, we will first smooth the segment at the edge of layer variation. For example, in Figs. 2-(c) and -(d), segments to be smoothed are marked with circles.

For the optimal rate adaptation, one difficulty is to estimate the statistics of the wireless link bandwidth accurately. It has been widely acknowledged that the Markov channel models are useful tools to describe the variations of wireless links [20], [21]. Thus, two discrete-time finite-state Markov models are employed for describing the available bandwidth of the WiFi and Bluetooth links, respectively. The Markov models for the channels can be obtained according to the recent measurements or using the history data. With the wireless channel model, we can derive the available bandwidth state transition probabilities. According to the Markov property, the state at any time instance only depends on its immediately previous state. Given any state s and action a, the transition probability of the MDP can be given as

$$\mathcal{P}^a_{ss'} = \Pr\{s_{n+1} = s' | s_n = s, a_n = a\}. \quad (1)$$

Considering two wireless links in our model, and assuming they are independent, the transition probability is calculated as

$$\mathcal{P}^a_{ss'} = \Pr\{bw' | bw\} \cdot \Pr\{bt' | bt\}. \quad (2)$$

With the above transition probability, for any state $s = \{q, \Delta q, v, \Delta v, t, \Delta t, bw, bt, d\}$, if action A_b is taken, then the next state $s' = \{q', \Delta q', v', \Delta v', t', \Delta t', bw', bt', d'\}$ can be derived as

$$\begin{aligned} q' &= q - \lceil SZ^{v'}_{d+1}/(bw' + bt') \rceil + 1, \\ \Delta q' &= q' - q, \quad v' = v + m, \quad \Delta v' = m, \\ \Delta t' &= SZ^{v'}_{d+1} \cdot bw'/(bw' + bt'), \\ t' &= t' + \Delta t', \quad d' = d + 1, \end{aligned} \quad (3)$$

where $SZ^{v'}_{d+1}$ is the size of segment $d+1$ at quality version v', and m is the action value. The video segment information can be obtained by the client during the initialization step. Similarly, it is easy to derive the next state with A_u, A_w and A_s actions.

TABLE I
REWARDS ASSOCIATED WITH STATES

State $s_t = s$	Reward $R(s)$
$(*, *, *, *, *, *, *, *, N_T)$	0
$(q, *, *, *, *, *, *, *, *)$, if $q < T_{qmin}$	$-(Q_L - q) + \Delta q$
$(q, *, *, *, *, *, *, *, *)$, if $q > T_{qmax}$	$-q - \Delta q$
$(*, \Delta q, *, \Delta v, t, \Delta t, *, *, *)$	$\min(-\lvert\Delta v\rvert, -\lvert\Delta q\rvert) - \Delta t R_t$
other states	0
any state with smooth action	R_s

In order to measure how good the actions are, we define a reward value r associated with each action a. The reward value r_n at step n is determined by the previous state s_{n-1}, *i.e.*, $r_n = R(s_{n-1})$. As shown in Table I, we define the reward values for different states. In the reward function table, $*$ means that the state can be of any value. The reward of a state will be looked up from the top to the bottom in Table I, until one entry is matched. When $d = N_T$, all the segments have been downloaded, and thus reward 0 is given. The reward functions are highly related to the streaming

QoS. A higher reward is more desirable so the corresponding action is preferable. Video playback freezes could be the most undesirable experience, and thus a negative reward with a large magnitude is given when q is less than the threshold T_{qmin}, as there is a high probability of playback freeze when the queue length is very small. Such a negative reward can also reduce the startup latency, since we prefer the actions to fill up the buffer with less time to avoid large negative rewards. Therefore, low-layer segments will be requested when the queue length is small. Similarly, we also give a negative reward with a large magnitude when q is larger than the threshold T_{qmax}, in order to avoid buffer overflow. When the queue length is between the two thresholds, we prefer less variation of q, so we set the reward to no larger than $-\Delta q$. To obtain a smooth and high quality video streaming, we also give negative rewards when there are high fluctuations of Δv. According to [22], we can assume that the cost function of WiFi $C_w(x)$ is a constant (flat fee) independent of the traffic load:

$$C_w(x) = R_w, \tag{4}$$

and the cost function of Bluetooth $C_t(x)$ is a constant plus a linear function of the usage

$$C_t(x) = K_t + R_t x. \tag{5}$$

Here, the cost function of Bluetooth is quite similar to the cellular data plan. (Other cost models can be considered, which are not included in this work due to space limit.) When the usage exceeds a cap K_t, then additional data will be charged at a higher rate. Therefore, when the cap is reached, the usage of Bluetooth traffic must be restricted. We give a negative reward equals the traffic cost $\Delta t R_t$ when additional Bluetooth data transfer is used. To achieve a more smoothed perceived streaming quality, we will assign additional reward R_s upon the invoking of the smooth action.

The streaming policy π is a mapping of the possible action at each step. The long-term reward $V^{\pi}(s)$ under policy π can be computed as:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{\sum_{n=0}^{N_T} r_n | s_n = s\right\} \\ &= \sum_{s'} \mathcal{P}^{a}_{ss'}[R^{a}(s) + \gamma V^{\pi}(s')], \end{aligned} \tag{6}$$

where R is the next reward of taking action a at state s, γ is the discount rate and $0 \le \gamma \le 1$. The parameter γ makes a trade-off between myopic video quality and future interruptions and variations. A small γ lets future reward weigh less, and thus makes the adaptation decision more myopically. Meanwhile, when fewer future steps are considered, it also results in a more myopic decision.

Obviously, finding the optimal strategy policy $\pi^*(s)$ which can maximize the long-term reward is the goal of the reinforcement learning task of video streaming. Thus, our multi-link video streaming task can be finally formulated as an optimization problem:

$$\pi^*(s) = \arg\max_{\pi} \sum_{s'} \mathcal{P}^{a}_{ss'}[R^{a}(s) + \gamma V^{*}(s')]. \tag{7}$$

IV. Practical Algorithm Design

Given the formulated video streaming process as an optimization problem, our goal is to find the best solution of the problem, which is also the optimal streaming policy. Theoretically, dynamic programming can be employed to solve the above optimization problem by value iteration. The computation time and the memory consumption of the dynamic programming algorithm are determined by the number of states. With about 50 segments, the computation time and the solution table may exceed one hour and 600 MBs on a high-end desktop, respectively. Therefore, this approach is not suitable for real-time adaptive streaming. To overcome this problem, we aim to develop a real-time best streaming action search algorithm to find a sub-optimal solution for the optimization problem formulated in the previous section.

A. Bandwidth Estimation

Rapid network load changes and short-term outages are difficult to predict, and the resultant available bandwidth for a session becomes a time-varying random process. Thus, instead of using a homogeneous Markov chain to estimate the available bandwidth, in our work, a heterogeneous and time-varying Markov model is used to estimate the future bandwidth. The bandwidth of each link will be divided into several regions. Each region will represent a state of the Markov channel model, and the total number of the states is equal to the number of regions. Assume that there are n states, then an $n \times n$ transition matrix P will be used for the Markov channel model. Each element p_{ij} is the transition probability from state i to j. To obtain the transition probability, another $n \times n$ matrix C is used to count the number of transitions for each state. Once a segment has been successfully downloaded, the transmission bandwidth can be calculated by dividing the total size of the data transmitted over the total transmission time. Then the bandwidth region can be determined and we will increase the corresponding c_{ij} by one. p_{ij} is updated by the following equation:

$$p_{ij} = \frac{c_{ij} + 1}{\sum_{j=1}^{n} c_{ij} + n}. \tag{8}$$

Initially, if there is no history data available, $c_{ij} = 0$, and p_{ij} is set to $1/n$. The transition matrix will be updated after each segment has been successfully downloaded, so the transition matrix can better predict the future bandwidth variations with the recent measurements.

B. Real-time Search Algorithm

According to the formulation of the video streaming process, the calculation of the best long-term reward at state s with action a in (6) can also be written as:

$$Q^{*}(s,a) = R(s) + \gamma \sum_{s'} \mathcal{P}^{a}_{ss'} V^{*}(s'), \tag{9}$$

where $V^{*}(s')$ is the best long-term reward for state s'. It is easy to note that the best long-term reward for the current state is determined by all the possible future states. Since dynamic programming considers all the possible future steps to obtain the optimal solution, it results in an extremely

Algorithm 1 Real-Time Best-Action Search Algorithm

```
procedure GETBESTACTION(s)
    Initialize action ← −1, Q_max ← −∞
    Generate all possible actions A(s) for state s
    for all Action a ∈ A(s) do
        q ← REWARDSEARCH(s, a, 0)
        if q > Q_max then
            Q_max ← q, action ← a
        end if
    end for
    return action
end procedure

procedure REWARDSEARCH(s, a, d)
    q ← reward of (s,a)
    if d ≥ D then
        return q
    end if
    Generate all possible next states S' of (s,a)
    for all s' from S' do
        Q_max ← −∞
        Generate all possible actions A'(s) for state s'
        for all Action a' ∈ A'(s) do
            Q_t ← REWARDSEARCH(s', a', d + 1)
            if Q_t > Q_max then
                Q_max ← Q_t
            end if
        end for
        q ← q + γ P_ss' Q_max
    end for
    return q
end procedure
```

long computation time. If only part of the future steps are considered, a sub-optimal solution can be obtained. Based on this idea, we develop a real-time recursive best-action search algorithm, which is shown in Algorithm 1.

To meet the requirement of the real-time search, an important issue is to reduce the search duration for each state to an acceptable value. We achieve this goal by setting a small search depth D to invoke the search algorithm. For the current state s, all the possible actions $\mathcal{A}(s)$ will be enumerated. The recursive reward search algorithm is invoked to obtain the reward of state s with action a by enumerating all the possible future states S' and their associated actions $\mathcal{A}'(s)$.

C. Adaptive Search Depth

Search depth is an important issue in our work. The search depth can determine how good the search result is, and a larger value of depth will achieve a better result. Meanwhile, with the increment of the search depth, the search time to obtain the action for a segment will be increased exponentially. Therefore, the search depth can be viewed as a trade-off between the video quality and the search time.

Based on several preliminary experiment results, when the search depth D is larger than three, it will take more than two seconds to obtain a decision on the test Android smart phone. Thus, the maximum search depth D_{max} is set to three. As the perceived video streaming fluency is generally considered as one of the most important QoS for the user, the search depth D is determined by the current queue length in our work. We divide the buffer queue into three regions, $[0, q_1)$, $[q_1, q_2)$ and $[q_2, q_L]$. For each state, the search depth D is determined according to its queue length q as follows:

$$D = \begin{cases} 1 & \text{if } q \in [0, q_1) \\ 2 & \text{if } q \in [q_1, q_2) \\ 3 & \text{if } q \in [q_2, q_L] \end{cases} \tag{10}$$

When the queue length is low, there is a high probability that a playback interruption may occur soon, and thus a short search time and depth is preferred. When the queue length is high, there is sufficient time to search a deeper depth to obtain a better result.

D. Discussion

According to [19], the MDP can be viewed as a decision tree. The current state represents the root of the decision tree, and the future possible actions and states form the node and leaves. Since the recursive search will not try the next action until it reaches the leaves. Thus, our real-time search algorithm is a depth-first algorithm. It is easy to find that the computational complexity of our real-time search algorithm is $O(b^D)$, where b is the total number of branches of the search tree and D is the search depth. If we only search one step, it is a typical greedy algorithm. When the search depth is equal to the total number of video segments N_t, then it is exactly identical to the dynamic programming algorithm. There is no need to store all the states and actions in the stack while searching the tree, so the memory consumption of our recursive search algorithm is not high. Generally, the space complexity of our algorithm is bound by $O(bD)$.

V. Performance Evaluation

A. Testbed Implementation

In order to better evaluate the performance of our proposed video streaming algorithm, we established a realistic testbed to measure the performance with a real video stream and an Android mobile phone.

A smart phone with Android OS is used as the client. The smart phone is integrated with WiFi, 3G and Bluetooth wireless interfaces. Since WiFi and 3G cannot work together to access the Internet on Android OS, WiFi and Bluetooth are utilized together to transmit the video segments simultaneously according to our multi-link streaming solution.

We implemented a video streaming application which can request the video contents from the web server through the HTTP/1.1 protocol. The video streaming application contains three main components: the streaming action search module, WiFi connection management module and Bluetooth connection management module. When the video streaming process begins, the streaming action search module will make the decision on how many enhancement layers to be requested and how to assign the transmission load to each link. Once the decision is made, WiFi and Bluetooth modules send HTTP/1.1 requests to the server to fetch the corresponding video segments. This process will continue until the last segment is successfully fetched. As there is no available SVC decoder for Android OS yet, we only use the smart phone to request the video segments and record the transmission trace.

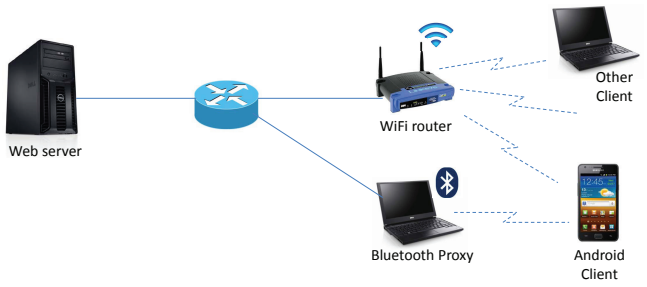


Fig. 3. Testbed Network Topology.

With the trace, we can decode the SVC video on a PC to evaluate the experiment results.

The video information is stored in a simplified manifest file in our implementation. In the manifest file, first the general video information is listed: the total number of segments, the duration of a segment, the total number of enhancement layers and the correspondent resolutions. Then the detailed bit-rate of each segment and the size of each layer for each segment are listed. The manifest file will be transmitted to the client first before the stream begins.

As Bluetooth cannot connect the client to the web server directly, we implemented a Bluetooth proxy to let the client access the web server. The Bluecove Java Bluetooth library is used to implement the proxy on a laptop. When the client sends an HTTP/1.1 request, the proxy will forward the request to the web server, fetch the video segments from the server and also forward the fetched segments to the client.

B. Experiment Settings

The network topology of our testbed is shown in Fig. 3. The video streaming server is deployed on a web server, which is deployed on a desktop by Apache HTTP Server 2.2.24 with Ubuntu 10.04 OS. Both the wireless router and the Bluetooth proxy are connected with the web server through wired links. The wireless router is flashed with OpenWrt to set up customized configurations. In our experiment, the WiFi network is configured to be in the IEEE 802.11b mode and the wireless transmission rate is set to 1 Mbps or 2 Mbps. The minimum round-trip time (RTT) of WiFi without queuing delay is about 13.19 ms. The Bluetooth proxy runs on a laptop which is equipped with a dual-core 2.53 GHz Intel CPU and 2 GB memory. We used a Samsung i9100 Galaxy II smart phone with Android 4.0.3 OS as the client. The Samsung i9100 is equipped with a dual-core 1.2 GHz Cortex-A9 CPU and 1 GB RAM. The minimum RTT between the smart phone and the proxy is about 25.83 ms, and between the proxy and the server is about 1.79 ms, respectively. Thus the total minimum RTT for the Bluetooth link is about 27.62 ms.

The open-source SVC codec JSVM [23] is used to encode a test video "Big Buck Bunny" (available at http://www.bigbuckbunny.org), and we did not pack the encoded video with any container. According to [24], the multi-segment strategy is adopted in our implementation. The test video is first chopped into small segments and then encoded into a base layer and two enhancement layers. In order to better test the feasibility and robustness of our approach, two different configurations of the encoded videos are employed, which are shown in Table II. The frame rate and Group of Pictures (GoP) of both configurations are set to 24 fps and 8 frames which include one I frame and seven hierarchically predicted B frame. To ensure that each segment contains complete GoPs, the length of the two configuration 1 and 2 are set to 17 frames (one independent I frame and two GoPs) and 49 frames (one independent I frame and six GoPs) per segment respectively. Therefore, the duration of each segment is about 700 ms and 2,042 ms for the two configurations, respectively. In our experiments, $N_t = 200$ segments are selected for configuration 1 and $N_t = 100$ segments for configuration 2.

TABLE II
VIDEO CODING CONFIGURATIONS

	Resolution	Avg. bit-rate (Kbps)	Std bit-rate deviation	Y-PSNR (dB)	Layer
Cfg 1	320×180	112.84	39.01	30.99	1
	320×180	238.94	88.84	32.63	2
	640×360	363.82	140.33	35.9	3
Cfg 2	640×360	235.4	92.09	35.37	1
	1280×720	531.1	215.97	38.53	2
	1280×720	1,056.9	469.1	41.5	3

At the client, the buffer size is set to $q_L = 20$ segments, which means that it can cache at most 20 video segments. T_{qmin} and T_{qmax} are set to 2 and 18 segments, respectively. For the Markov channel model, the available bandwidth for each link is divided into four regions, and thus two 4×4 transition matrices were used for the WiFi and the Bluetooth links, respectively.

To reduce the search time of our real-time best-action search algorithm, we implemented it in a non-recursive way. According to the experiments, we found that the running time of the algorithm is about 2 ms when the depth $D = 1$, 42 ms when $D = 2$, and 513 ms when $D = 3$, respectively. Therefore, we set the adaptive searching depth region as $q_1 = 8$ segments and $q_2 = 15$ segments.

C. QoS Metrics

Several objective tests were conducted to evaluate the performance of the proposed approach using several QoE related QoS metrics. For instance, according to [25], the layer variations will decrease the users' watching experience, and thus the number of layer variations was used to reflect the smoothness in our experiments. The performance metrics used in our experiments are listed as follows:

- **Startup latency (SL)**: It is defined as the duration from sending out the first segment request to the beginning of playing back the first segment, which can be also regarded as buffering time.
- **Playback fluency**: The number of segments that miss the playback deadline, and the playback freeze ratio, which calculates the percentage of the duration of playback freezes over the total video streaming time are evaluated.
- **Average playback quality (APQ)**: The average PSNR of all received frames.
- **Layer variation**: To measure how smooth the perceived video quality is, we count the number of layer variations.

Besides, the Bluetooth traffic usage was also utilized as one evaluation metric.

D. Experiment results

The available bandwidth in the wireless network is an uncontrollable, random process. In order to make the experiments reproducible, the traces of the wireless bandwidth were recorded (available at http://www.ece.uvic.ca/~mxing/mdp/bdtraces.mat), and a Linux traffic shaping command tool tc is utilized to shape the bandwidth based on the traces. For each scenario, the experiment is repeated 10 times to obtain the average. Since the commercial DASH solutions are not open-sourced, it is difficult to modify them to support multi-link and SVC. Thus, we compared our algorithm (RTRA) with the state-of-the-art one, the rate adaptation video streaming algorithm by K. Evensen (KERA) published in [18]. The only difference of the implemented KERA algorithm is that we modified it to support SVC videos. We tested the single-link case by enabling only one link during the experiments for both algorithms (RTRA_S and KERA_S) as well, which can also validate the benefit of multiple links.

1) Slow-changing Bandwidth Scenario: First, we investigate the performance with the slow-changing bandwidth scenario. To generate the bandwidth variation, we added slow-changing on/off background traffic by letting another laptop request a large file and then letting it sleep for 10 seconds. This procedure will repeat during the whole experiments. The maximum WiFi rate is set to 1 Mbps for the first video configuration and 2 Mbps for the second one, respectively.

The parameter α is set to one to make a good trade-off between the smoothness and video quality. As the total sizes of all the video segments of the first configuration are 1.9 MB for the base layer, 2.13 MB for the first enhancement layer, and 2.1 MB for the second enhancement layer, we set the maximum threshold of the Bluetooth traffic to $BT_t = 2$ MB to limit the overuse of Bluetooth traffic. Similarly, the $BT_t = 5$ MB is set for the second configuration video.

The results of the first set of experiments are shown as Case 1 in the first and fourth rows of Table III. We can notice that the proposed RTRA needs significantly less buffering time than KERA under both configurations. As a low buffer level brings a large-magnitude, negative reward which is undesired, the proposed RTRA will only request the base layer for the first several segments to quickly fill the buffer. KERA does not have such a low-layer start mechanism, so it may request high-layer video segments at the beginning, which results in a longer SL. Even with a single link, RTRA_S still only experiences almost a half SL when compared with that of KERA_S. Without the Bluetooth link, slightly longer SLs for both algorithms are observed.

As the aggregated bandwidth is sufficient to support the base layer for both video configurations, there is no segment missing the playback deadline for both algorithms with two links, and therefore 0% playback freeze ratio is achieved as well. Even with a single link, the proposed RTRA_S can still avoid playback freeze. By giving a large-magnitude negative reward to the low buffer level, the proposed RTRA can maintain a relatively high buffer level, which guarantees the playback fluency as long as the average bandwidth can support the base layer. For KERA_S, the sudden bandwidth decrease makes the bandwidth estimation inaccurate. Thus, when there is no other link can compensate for the bandwidth, the low bandwidth can no longer support the high quality video and playback freeze happens.

The proposed RTRA can achieve similar perceived video quality as KERA. With the low-layer startup, the buffer can be filled very quickly. When the buffer level reaches a certain level, drastic buffer level increment will bring a negative reward, and thus the video quality will be upgraded to avoid such a penalty. In this way, high quality video streaming can be guaranteed. As KERA is quite greedy, it always tries to request the highest possible layer, and KERA can achieve the similar quality as RTRA.

Since layer change may bring a negative reward, the proposed RTRA tends to maintain the current video quality. Thus, the average number of layer variations for RTRA is quite small. As mentioned before, KERA is a greedy algorithm. One of the drawbacks of the greedy algorithm is that the smoothness is ignored as it only considers the instantaneous video quality. Thus, there is a significant number of layer variations for KERA. With a single WiFi link, as WiFi cannot sustain the first enhancement layer, sometimes it has to switch from the first enhancement layer to base layer in order to avoid playback freeze, which results in more layer variations for RTRA_S than RTRA.

Furthermore, with the cap of Bluetooth traffic, RTRA uses much less Bluetooth traffic than KERA which can avoid the additional cost and negative reward.

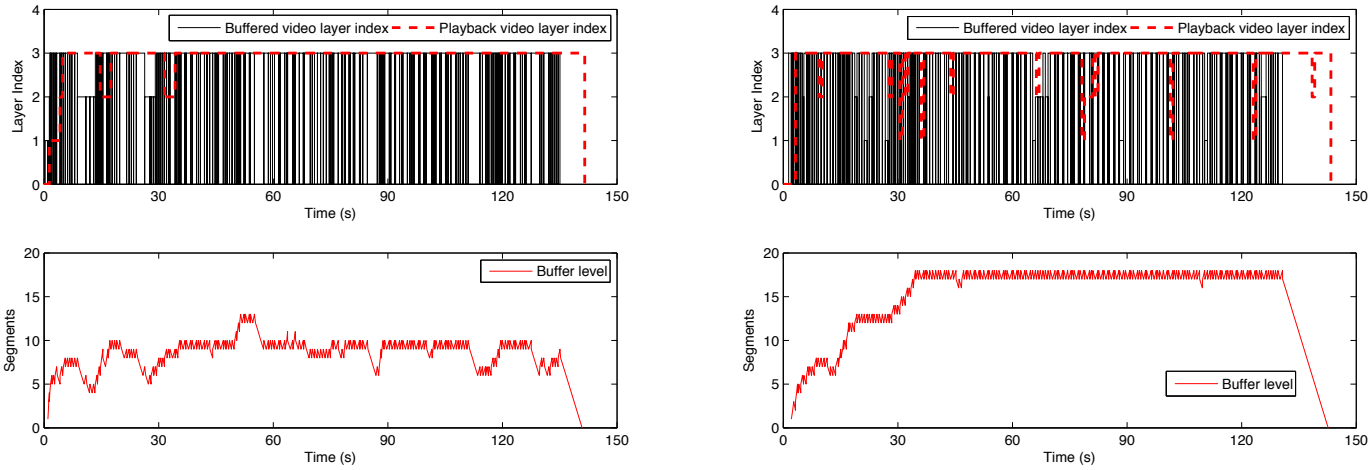
2) Rapid-changing Bandwidth Scenario: We then evaluate the performance with the rapid-changing on/off background traffic sharing the WiFi link with a much shorter on or off duration. Two different cases are generated: 1) We let another laptop request a file (600 KB) and then sleep for only 1 second before repeating the requesting procedure. In this way, there are some positive spikes of the available bandwidth in the WiFi link. 2) To generate the negative spikes for the available bandwidth in the WiFi link, we let the laptop request a small file (100 KB) and then sleep for 10 seconds. We conducted the two sets of experiments under the above two types of on/off background traffic.

In these two sets of experiments, the purpose is to evaluate the performance of the proposed RTRA with short-term bandwidth variations. We show the experiment results in the case 2A and case 2B rows of Table III for the positive and negative spikes cases, respectively. For the positive-spike case, the base layer cannot be sustained only with WiFi, so there are many segments missing the playback deadline under both video configurations for both algorithms with single link. When the second link is available, it is sufficient to support the base layer, therefore no segment missed the playback deadline and zero playback freeze ratio.

Under the positive-spike traffic, as the average bandwidth can support the base layer only, the RTRA algorithm tried to stay with the base layer. Even when the bandwidth was increased dramatically, quickly increasing the buffer level will bring a higher reward than improving the video quality by requesting more enhancement layers. Therefore, the smoothness

TABLE III
EXPERIMENT RESULTS

			SL (ms)	# of missed segments	Playback freeze ratio	PSNR (dB)	# of layer variation	BT traffic (KB)
Cfg 1	Case 1	RTRA	1,404	0	0%	34.07	19	1,624
		RTRA_S	2,184	0	0%	31.9	48.1	0
		KERA	2,718	0	0%	33.78	109	1,613
		KERA_S	3,144	26.1	5.36%	33.05	141.4	0
	Case 2A	RTRA	1,447	0	0%	32.19	40	1,687
		RTRA_S	2,393	56	14.4%	31.02	26.6	0
		KERA	3,215	0	0%	31.64	90	1,855
		KERA_S	3,661	98.7	21.63%	31.44	41.8	0
	Case 2B	RTRA	1,360	0	0%	35.72	5.8	1,136
		RTRA_S	1,597	0	0%	34.11	18.4	0
		KERA	1,985	0	0%	35.46	35.5	1,390
		KERA_S	2,744	0	0%	34.92	49.7	0
Cfg 2	Case 1	RTRA	3,573	0	0%	38.17	18.8	2,047
		RTRA_S	3,635	0	0%	36.51	31.8	0
		KERA	5,409	0	0%	38.24	59.9	2,466
		KERA_S	6,204	0.6	0.18%	37.73	57.6	0
	Case 2A	RTRA	5,131	0	0%	36.56	36.3	2,273
		RTRA_S	6,046	18.3	10.71%	35.64	13.2	0
		KERA	5,433	0	0%	36.57	42	2,597
		KERA_S	6,697	25.4	13.49%	36.0	24.5	0
	Case 2B	RTRA	2,505	0	0%	40.16	14.8	2,182
		RTRA_S	2,644	0	0%	39.17	17.4	0
		KERA	5,534	0	0%	40.34	31.6	2,256
		KERA_S	5,778	0	0%	39.95	38.7	0
	Robustness Case 2A/B	RTRA	4,023	0	0%	36.3	30	2,213
		RTRA	2,306	0	0%	40.37	7.1	2,102



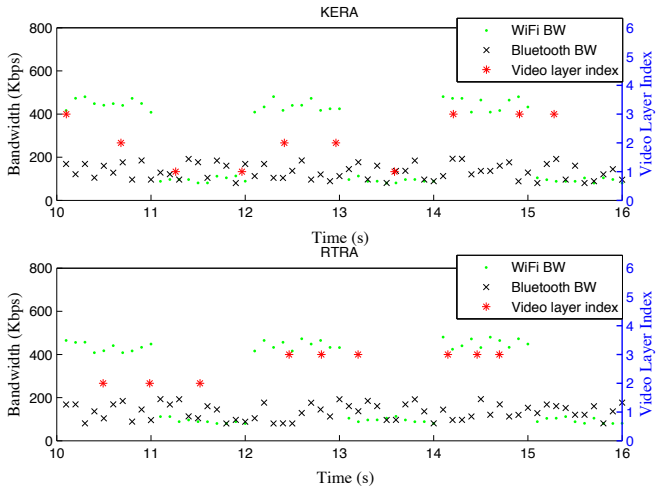
(a) RTRA (b) KERA

Fig. 4. Playback Traces and Buffer Occupancy Traces.

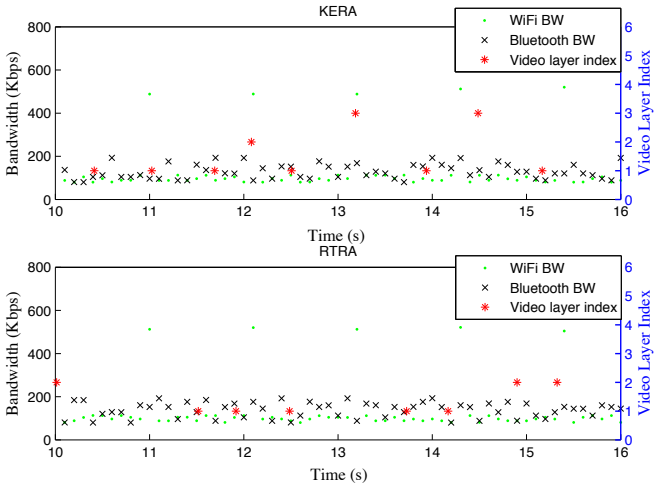
can be guaranteed. While under the negative spike traffic, the buffer will be quickly filled to reach a certain stable level. More enhancement layers will be requested to improve the video quality with a low playback freeze probability. The sudden bandwidth decreases do not have much impact on the video request decision, as the buffer has accumulated enough segments to maintain the current quality. KERA is very sensitive to the bandwidth variation, therefore it may switch to the base layer only during the negative spikes. As a result, compared to RTRA, a much lower smoothness can be achieved by KERA, no matter with one or two links.

To further demonstrate the performance of the proposed approach, we select one run of the experiment of case 2B and show the playback traces and buffer occupancy status of the two algorithms in Fig. 4. In the top sub-figures, the black rectangles represent the requested segment layer index, and the dashed curves represent the segment playback index. There are only several layer variations at the beginning of the streaming process. With the updated Markov state transition matrix, the prediction of future bandwidth becomes more accurate. When the buffer level reaches a certain level, RTRA can stay with the second enhancement layer all the time to obtain a high smoothness. In Fig. 4-(b), there are lots of layer variations during the whole streaming process. We can also notice that the buffer was almost full from the thirty-first second. With the almost full buffer, those layer variations indicate substantial bandwidth wastage.

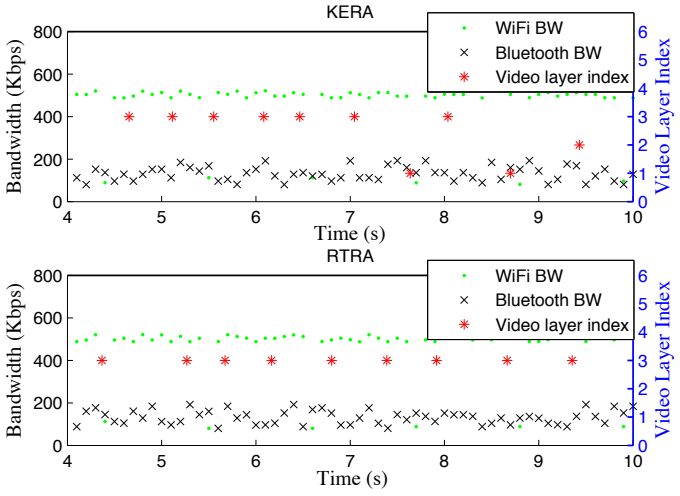
Figure 5 compares the video layer and bandwidth traces



(a) Case 1



(b) Case 2A



(c) Case 2B

Fig. 5. Comparison of Experiment Traces.

using KERA and the proposed RTRA under the first video configuration. All three sub-figures clearly show the sensitivity of KERA with regard to the variation of bandwidth. Layer variations occur frequently with KERA, which is along with

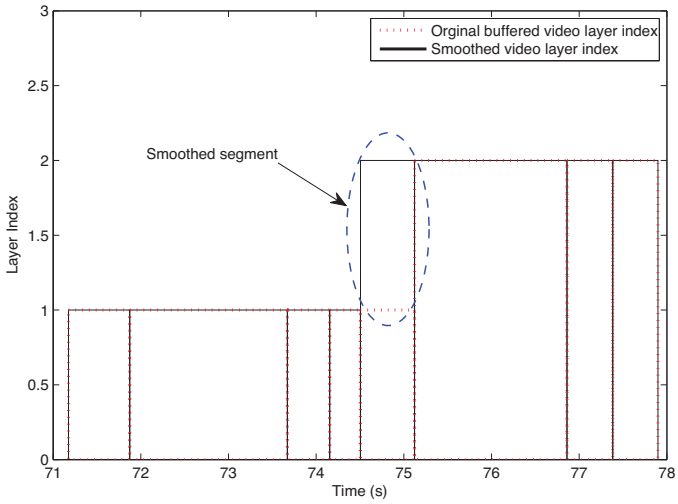


Fig. 6. Effect of The Smooth Action.

the bandwidth variation. Different from KERA, the proposed RTRA only requests the second layer unless the buffer occupancy is accumulated to a certain level before it requests the higher layer, which is shown in Fig. 5-(a). As there may be negative reward for the layer variation, it enables RTRA being immune to the bandwidth spikes, which can be found from Figs. 5-(b) and -(c).

For RTRA, when the buffer level reaches a certain level, the smooth action will become available. By taking the advantage of SVC coding, we split the video segments into layers. Thus the enhancement layer can be requested individually to improve both the smoothness and quality. In Fig. 6, the effect of the smooth action is shown. The segment in the blue circle is smoothed by requesting the next enhancement layer.

3) Robustness Evaluation: Finally, we evaluate the robustness and the effectiveness of the proposed RTRA algorithm based on the Markov channel model. In the previous experiments, the initial state transition probability between any two states is set to be equal. With the recorded bandwidth traces, we can process the traces and obtain the state transition rate of the traces. Then, in the following experiment, we used the state transition rate to initialize the state transition probability matrix. The matrices are used as the initial matrix in our experiment. The experiment results are shown in the last row of Table III.

Since the initial state transition probability matrix in this experiment can accurately reflect the bandwidth variations, the results outperforms the previous results with a very limited margin in every aspect. In the previous practical approach, although the initial state transition matrix is not accurate, by updating it with the measured throughput, we still can achieve a satisfactory performance. Meanwhile, this also confirms the robustness of the proposed algorithm, as the accurate initial state transition matrix is not essential.

VI. Conclusions

In this paper, we proposed a real-time adaptive best-action search algorithm for video streaming over multiple wireless access networks. First, we formulated the video streaming

process as an MDP. To achieve smooth video streaming with high quality, we carefully designed the reward functions. Second, with the proposed rate adaptation algorithm, we can solve the MDP to obtain a sub-optimal solution in real time. Last, we implemented the proposed algorithm and conducted realistic experiments to evaluate its performance and compare it with the state-of-the-art algorithms. The experiment results showed that the proposed solution can achieve a lower startup latency, higher video quality and better smoothness.

There are still many open issues to investigate in the future. First, how to better allocate the loads between several links with finer granularity should be investigated. Second, to better predict the future bandwidth, the most recent estimation of bandwidth should be assigned with a higher weight. Last but not least, the size of the video segment should be further considered for variable bit rate (VBR) videos to improve the bandwidth estimation accuracy.

REFERENCES

[1] T. Stockhammer, "Dynamic adaptive streaming over HTTP –: standards and design principles," in *ACM MMSys'11*, 2011, pp. 133–144.

[2] K. Tappayuthpijarn, T. Stockhammer, and E. Steinbach, "HTTP-based scalable video streaming over mobile networks," in *IEEE ICIP'11*, 2011, pp. 2193–2196.

[3] R. Mok, X. Luo, E. Chan, and R. Chang, "QDASH: a QoE-aware DASH system," in *ACM MMSys'12*, 2012, pp. 11–22.

[4] S. Xiang, L. Cai, and J. Pan, "Adaptive scalable video streaming in wireless networks," in *ACM MMSys'12*, 2012, pp. 167–172.

[5] C. Mueller, S. Lederer, and C. Timmerer, "A proxy effect analyis and fair adatpation algorithm for multiple competing dynamic adaptive streaming over HTTP clients," in *IEEE VCIP'12*, 2012, pp. 1–6.

[6] T. Kupka, P. Halvorsen, and C. Griwodz, "Performance of on-off traffic stemming from live adaptive segmented HTTP video streaming," in *IEEE LCN'12*, 2012, pp. 401–409.

[7] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptive video players over HTTP," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 271–287, 2012.

[8] S. Xiang, "Scalable Video Transmission over Wireless Networks," Ph.D. dissertation, University of Victoria, 2013.

[9] L. Cai, S. Xiang, Y. Luo, and J. Pan, "Scalable modulation for video transmission in wireless networks," *IEEE Trans. Veh. Technol.*, vol. 60, no. 9, pp. 4314–23, 2011.

[10] S. Xiang and L. Cai, "Transmission control for compressive sensing video over wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 3, pp. 1429–37, 2013.

[11] M. Kobayashi, H. Nakayama, N. Ansari, and N. Kato, "Robust and efficient stream delivery for application layer multicasting in heterogeneous networks," *IEEE Trans. Multimedia*, vol. 11, no. 1, pp. 166–176, 2009.

[12] R. Zhang, R. Ruby, J. Pan, L. Cai, and X. Shen, "A hybrid reservation/contention-based mac for video streaming over wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 3, pp. 389–398, 2010.

[13] T. Luan, L. Cai, J. Chen, X. Shen, and F. Bai, "Engineering a distributed infrastructure for large-scale cost-effective content dissemination over urban vehicular networks," *IEEE Trans. Veh. Technol.*, 2013.

[14] A. Yaver and G. Koudouridis, "Utilization of multi-radio access networks for video streaming services," in *IEEE WCNC'09*, 2009, pp. 1–6.

[15] D. Kaspar, K. Evensen, P. Engelstad, and A. Hansen, "Using HTTP pipelining to improve progressive download over multiple heterogeneous interfaces," in *IEEE ICC'10*, 2010, pp. 1–5.

[16] K. Evensen, T. Kupka, D. Kaspar, P. Halvorsen, and C. Griwodz, "Quality-adaptive scheduling for live streaming over multiple access networks," in *ACM NOSSDAV'10*, 2010, pp. 21–26.

[17] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. Hansen, and P. Engelstad, "Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks," in *ACM MMSys'11*, 2011, pp. 57–69.

[18] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. F. Hansen, and P. Engelstad, "Using bandwidth aggregation to improve the performance of quality-adaptive streaming," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 312–328, 2012.

[19] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998, vol. 28.

[20] H. Wang and N. Moayeri, "Finite-state Markov channel-a useful model for radio communication channels," *IEEE Trans. Veh. Technol.*, vol. 44, no. 1, pp. 163–171, 1995.

[21] Q. Zhang and S. A. Kassam, "Finite-state Markov model for rayleigh fading channels," *IEEE Trans. Commun.*, vol. 47, no. 11, pp. 1688–1692, 1999.

[22] X. Hou, P. Deshpande, and S. Das, "Moving bits from 3G to metro-scale WiFi for vehicular network access: An integrated transport layer solution," in *IEEE ICNP'11*, 2011, pp. 353 –362.

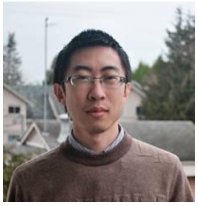
[23] J. Reichel, H. Schwarz, and M. Wien, "Joint scalable video model 11 (JSVM 11)," *Joint Video Team, Doc. JVT- X*, 2007.

[24] I. Kofler, R. Kuschnig, and H. Hellwagner, "Implications of the ISO base media file format on adaptive http streaming of h. 264/svc," in *IEEE CCNC'12*, 2012, pp. 549–553.

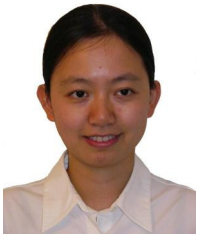
[25] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen, "Flicker effects in adaptive video streaming to handheld devices," in *ACM MM'11*, 2011, pp. 463–472.

[26] M. Xing, S. Xiang, and L. Cai, "Rate adaptation for video streaming over multiple wireless access networks," in *IEEE Globecom '12*, 2012, pp. 5745-5750.

Min Xing is currently a Ph.D candidate in the Department of Electrical and Computer Engineering, University of Victoria, British Columbia, Canada. He eceived B.S. degree in Computer Science from Soochow University, Suzhou, Jiangsu in 2007, and M.S. degree in Software Engineering from Tongji University, Shanghai, China in 2010. His current research interest is multimedia over wireless networks.



Siyuan Xiang (S'10-M'13) received the M.Eng. degree from Tongji University, Shanghai, China, in 2008, and the Ph.D. degree in Electrical and Computer Engineering from the University of Victoria, Victoria, BC, Canada, in 2013. His research interests include multimedia communications over wired and wireless networks.



Lin Cai (S'00-M'06-SM'10) received her M.A.Sc. and PhD degrees (awarded Outstanding Achievement in Graduate Studies) in electrical and computer engineering from the University of Waterloo, Waterloo, Canada, in 2002 and 2005, respectively. Since 2005, she has been an Assistant Professor and then an Associate Professor with the Department of Electrical & Computer Engineering at the University of Victoria. Her research interests span several areas in wireless communications and networking, with a focus on network protocol and architecture design supporting emerging multimedia traffic over wireless, mobile, ad hoc, and sensor networks.

She has been a recipient of the NSERC Discovery Accelerator Supplement Grant in 2010, and the best paper awards of IEEE ICC 2008 and IEEE WCNC 2011. She has served as a TPC symposium co-chair for IEEE Globecom'10 and Globecom'13, and the Associate Editor for IEEE Transactions on Wireless Communications, IEEE Transactions on Vehicular Technology, EURASIP Journal on Wireless Communications and Networking, International Journal of Sensor Networks, and Journal of Communications and Networks (JCN).