Implementation and Evaluation of Adaptive Video Streaming based on Markov Decision Process

Ayub Bokani, S. Amir Hoseini, Mahbub Hassan, Salil S. Kanhere
School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia
Email: {abokani, amirhoseini, mahbub, salilk}@cse.unsw.edu.au
Data61|CSIRO

***Abstract*—In HTTP-based adaptive streaming systems, media server simply stores video content segmented into a series of small chunks coded in different qualities and sizes. The decision for next chunk's quality level to achieve a high quality viewing experience is left to the client which is a challenging task, especially in mobile environment due to unexpected changes in network bandwidth. Using computer simulations, previous work has demonstrated that Markov decision process (MDP) is very effective for such decision making and that it can reduce video freezing or re-buffering events drastically compared to other methods of adaptation. However, to date there has been no practical implementation and evaluation of MDP-based DASH players. In this work, we extend a publicly available DASH player recently released by DASH industry forum to realise a real DASH player that implements MDP-based video adaptation. We implement two alternative MDP optimisation algorithms, value iteration and Q learning and evaluate their performances in real driving conditions under 300 minutes of video streaming. Our results show that value iteration and Q learning reduce video freezing by a factor of 8 and 11, respectively, compared to the default decision making algorithm implemented in the public DASH player.**

I. Introduction

To achieve scalable distribution of rapidly growing video contents, there is a strong push from the industry to adopt HTTP as a universal platform for delivering all types of contents, including video. Apple [1], Microsoft [2], and Adobe [3] have already deployed their own proprietary HTTP-based video streaming platforms while a standard, called dynamic adaptive streaming over HTTP (DASH) [4], has been introduced by the world wide web consortium (W3C) to facilitate wide-spread deployment of this technology. Recently, a DASH industry forum [5] has been formed to catalyse the development, testing, and adoption of DASH technology.

The key concept in DASH is to code the same video in multiple bitrates (qualities) and store each stream as a series of small video chunks of 2-10 sec duration. A client simply downloads and plays a chunk of a given quality using the standard HTTP GET command used for fetching any other objects on the Web. Since video has strict display deadlines for every frame, each chunk needs to be downloaded before its deadline to avoid the 'freezing' effect. It therefore becomes the responsibility of a DASH player to dynamically select the 'right' quality of the next chunk to ensure a smooth video at the receiver with the highest possible quality and minimum number of quality switches from one chunk to the next.

For a DASH player, the quality selection for the next video chunk is a challenging problem, especially under vehicular environment with significant uncertainty in cellular bandwidth as the player rapidly changes its location. Using computer simulations, previous works [6]–[8] have demonstrated that Markov decision process (MDP) is very effective for making such video adaptation decisions and that it can reduce video freezing events drastically compared to other methods of adaptation. However, to date there has been no practical implementation and evaluation of MDP-based DASH players. In this work, we set out to empirically evaluate the effectiveness of MDP-based DASH in vehicular environments.

The contributions and novelty of the paper can be summarised as follows:

- To the best of our knowledge, we realize the first DASH player that uses MDP to make its quality selection decisions. We achieve this by extending a Javascript-based publicly available DASH player released by DASH industry forum [5].
- We implement two alternative MDP optimisation algorithms, *value iteration* and *Q learning* and evaluate their performances using an Android phone under real driving conditions with 300 minutes of video streaming. Our results show that value iteration and Q learning reduce video freezing by a factor of 8 and 11, respectively, compared to the default decision making algorithm implemented in the public DASH player.

The rest of the paper is structured as follows. In Section II, we provide a review of MDP and its optimization algorithms as applied to DASH. Section III explains the extension of industry forum's DASH player to implement the MDP algorithms. Experimental setup is presented in Section IV, followed by evaluation results in Section V. We conclude the paper in Section VI.

II. Review of MDP-based DASH

MDP is a *well-known* reinforcement learning technique to solve problems under uncertainty. An MDP is a tuple $(S, A, P_{sa}, R_{sa}, \gamma)$, where S is a set of *states*, A is a set of *actions*, P_{sa} is a transition probability distribution over the state space when action a is taken in state s, R_{sa} is the immediate *revenue* for taking action a in state s, and $\gamma \in [0, 1)$ is a discounting factor for the revenues collected from future actions and states. The solution of the MDP is a policy that

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

tells us how to make a decision, i.e., choose an action when a particular state is observed. There are many possible policies, but the goal is to derive the *optimal policy* that maximizes the expected revenue by considering the immediate as well as the future discounted revenues. How we obtain the MDP parameters for a DASH client is explained in our previous works [6], [7] with all details. In this work, we consider two different methods to derive the optimal policy, namely value iteration and Q-learning. They have different strengths and weaknesses and our goal is to compare their performances in the context of DASH quality decision making.

Value iteration assumes *a priori* knowledge of state transition probabilities and using it to derive the optimal policy before the system starts its operation. On the other hand, Q-learning assumes no *a priori* knowledge of transition probabilities, but *learns* the optimal policy on-line as actions are taken and states are observed. In essence, these two methods allow trading *learning time* with *a priori* knowledge. We briefly introduce these two methods as follows.

A. Value Iteration

The core idea is to use the transition probabilities to work out the future states and then calculate the expected total revenue or *value* $V(s,a)$, i.e., for a given action a taken in a given state s [9]. The action that provides the maximum revenue for a given state s is selected as the optimal action for that state. Optimal actions for all states then constitute the optimal policy. The values can be computed using the following algorithm that repeats the process until it converges, i.e., no value change by more than a small number δ:

1) For each state-action pair, initialize $V(s,a) = 0$.
2) Repeat until convergence:

$$V(s,a) = R(s,a) + max\gamma \sum_{s} P(s,a,s')V(s',a'). \tag{1}$$

As we discussed in [7], the transition probabilities for DASH are basically obtained from the bandwidth CDF. Value iteration, therefore, requires *a priori* knowledge of bandwidth CDF, which may or may not be available to a DASH client.

B. Q-learning

In this technique [10], a Q matrix which defines the values of every state s if action a is taken is initialized with zero. Whenever the DASH client starts out in state s, takes action a, and ends up in state s', it updates $Q(s,a)$ as:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma max_{a'}Q(s',a')] \tag{2}$$

where α is the learning rate. When state s is observed, the Q matrix, or the particular row in the matrix for state s, is used to make the decisions in the following way: choose the action that provides the maximum value based on the current estimates in Q for *most of the time*, but a *random action the rest of the time*. We use the *Boltzmann distribution* function for this probabilistic decision making purpose in the following way:

$$f(s,a) = \frac{e^{Q(s,a)/\theta}}{\sum_j e^{Q(s,a_j)/\theta}} \tag{3}$$

where $f(s,a)$ is the probability of selecting action a when in state s, and θ, often referred to as temperature, controls the degree of randomness in choosing an action. With large θ, actions will be fairly randomly selected irrespective of the values in Q, but for a small θ closer to zero, the best action based on Q values will be selected with high probabilities. For a DASH client with no *a priori* knowledge of the bandwidth, it is useful to begin with a large θ so the client makes mostly random decisions and learns from its observations. As it continues to learn, the Q values are used more often to make the decisions.

III. Implementation of MDP-based DASH Players

A. DASHIF Player

The DASH Industry Forum introduced a *JavaScript* DASH player in which the optimization is done based on monitoring the available bandwidth as well as buffer occupancy level [5] at the client. In this paper, we have modified the quality selection module of this player in different ways to employ our MDP-based rules. In the following, we briefly explain how the *basic* DASH player works followed by explaining our modifications in new players.

The algorithm used by DASH Reference Player considers two parameters for making decisions on bitrate adaptation: *Buffer level* and possible *download ratio*. In this player, a download ratio μ is computed by dividing the video chunk duration *VCD* by its fetching time *CFT* multiplied by a constant called safety factor (ϕ) as:

$$\mu = \frac{VCD}{CFT} \times \phi \tag{4}$$

The safety factor ϕ which can vary between 0 and 1 is set to 0.75 by default to ensure that player behaves more conservatively and there is sufficient bandwidth before initiating the *switch up* action. If the resulting μ is greater than 1, then the algorithm will decide to switch up to a higher bit rate. Otherwise it will initiate the *switch down* action. In addition, the player has to avoid an interruption to the stream (i.e. deadline miss). Thus regardless of the download ratio, if the buffer level is less than a certain threshold, the player will chose to take the *switch down* action. To get better insights of the basic DASH player, we briefly discuss its responsible components for controlling the bitrate adaptation as follows.

- *BaseRulesCollection.js*: The main controlling file which tells which rules to execute (Listing. 1). There are three main rules namely *downloadRatioRule*, *insufficientBufferRule* and *limitSwitchesRule*.
- *DownloadRatioRule.js*: Main quality adaptation controller that works based on aforementioned algorithm.
- *InsufficientBufferRule.js*: A strict rule to prevent buffer drains. In this rule the minimum buffer level can be set using *DryBufferlimit* variable. The variable

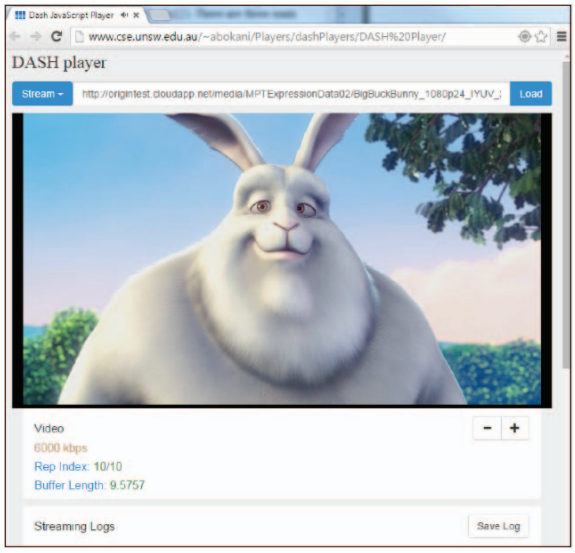


Fig. 1: DASH Industry Forum's Player

dryBufferHits is responsible for counting the number of bufffer drains (i.e., deadline miss).

- *limitSwitchesRule.js*: To control the number of quality switches which is disabled by default.
- *BufferExtensions.js* : Using variables *minBufferTime* and *bufferTime* the maximum buffer length can be adjusted here. We will use this rule to adjust the maximum buffer length of this player to our new DASH players.

Fig. 1 illustrates a screen-shot of DASHIF player. As it's shown, the *Save Log* button is added to the player's interface to record the streaming logs for evaluation purpose. The time that a video chunk is fully received, the buffer level on that time, the selected quality level as well as buffer drain and quality change events are all stored in this log file. We will use the quality level of all video chunks to calculate the average quality (AQ) over a streaming session. The buffer drained and quality change events will be also used to measure the total number of deadline misses (DM) and quality changes (QC) respectively.

B. MDP-based DASH Players

We created MDP-VI and MDP-QL based DASH players by changing the bitrate adaptation *rule* of the DASHIF player as follows. First, we wrote two new rules, *mdpVIRule.js* and *mdpQLRule.js* based on Value Iteration and Q-learning algorithms respectively. Then, we modified the *BaseRulesCollection.js* by including our new rules and disabling all the other existing rules (Listing. 2). Our *mdpVIRule.js* class monitors the current system state and using a pre-stored policy table selects the corresponding quality level. With *mdpQLRule.js* class, after receiving each video chunk the values of θ and Q-table are updated and next chunk's quality is selected based on these values. These rules are stored in DASHIF's streaming rules folder (i.e., ../app/js/streaming/rules/).

IV. Experiment Set-up

In this section we explain different settings and experiment setup. The DASHIF player (non-MDP) which is used as benchmarking model in this paper, does not require background information or training phase. Our MDP-based DASH players however, require bandwidth model (MDP-VI) and a learning process (MDP-QL) prior to experiments which are explained.

Listing 1: DASHIF player

```
MediaPlayer.rules.BaseRulesCollection =
    function () {
 2  "use strict";
 3  var rules = [];
 4  return {
 5    downloadRatioRule: undefined,
 6    insufficientBufferRule: undefined,
 7    //limitSwitchesRule: undefined,
 8    getRules: function () {
 9       return Q.when(rules);
10    },
11    setup: function () {
12      var self = this;
13      self.getRules().then(
14        function (r) {
15          //r.push(self.mdpRule);
16          r.push(self.downloadRatioRule);
17          r.push(self.insufficientBufferRule);
18          //r.push(self.limitSwitchesRule);
19        } );
20    } } };
```

Listing 2: MDP-VI based DASH player

```
5  ...
6   return {
7      mdpVIRule: undefined,
8      //mdpQLRule: undefined,
9      //downloadRatioRule: undefined,
10     //insufficientBufferRule: undefined,
11     //limitSwitchesRule: undefined,
12     getRules: function () {
13        return Q.when(rules);
14     },
15 ...
```

A. MDP-VI DASH Player

As discussed in [6], [7] and [11], to generate the MDP-VI's streaming policy, a sufficient amount of bandwidth samples are required to satisfy the normal distribution assumption. For this purpose, we conducted a comprehensive data collection campaign on real-world 3G network, on particular route that we chose for the experiments. Fig. 2 illustrates the NICTA-UNSW Sydney route (4.7 km - around 15 minute drive) that was selected for our experiments, data collection and testing the DASH players.

1) Bandwidth measurement application: Although it might make sense to utilise one or multiple of existing measurement and network coverage systems such as OpenSignal [12] or NetIndex [13] to obtain the desired data, use of these systems comes with a cost. An alternate approach is to develop multiple applications and components which will be combined to form

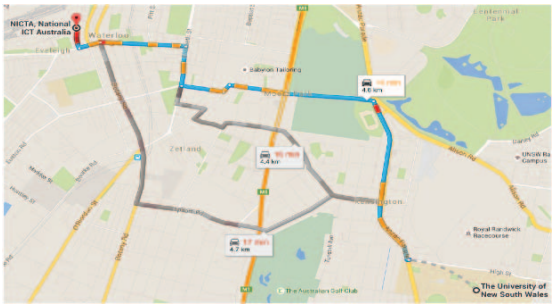


Fig. 2: NICTA-UNSW route: 4.7 km - 15 minute average driving time

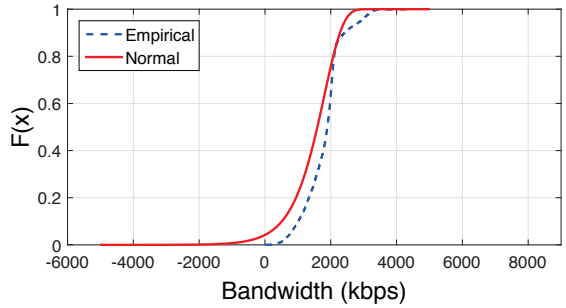


Fig. 3: Empirical Vs Normal CDFs of bandwidth samples

the final system. Building an entirely new system will provide flexibility in design and for future extensibility as it is not limited to the capabilities of the existing systems. We managed to create a user friendly *Android* application in which only a single button is required to be clicked to measure the available bandwidth (bw) and store it in the initialized Parse server [14]. This is done by downloading a file and dividing its size S_f to its fetching time T_f (i.e., $bw = \frac{S_f}{T_f}$).

2) Dataset: We stored an image file of 500 Kb size on our UNSW/CSE web server and using our Android application downloaded it every 10 second to measure the available bandwidth. This is a typical approach employed by speed test applications. During the whole measurement process, two Android 3G mobile phones were conducted to speed up the data collection. In total, 32 return trips were made and 9960 bandwidth samples were recorded. Fig. 3 demonstrates the normal and empirical bandwidth CDFs ($\mu = 1802.8$ kbps $\sigma = 572.77 kbps$). As it's shown in our dataset, bandwidth is distributed close to normal.

Next, we use this data as input to our MDP-VI model and solve it offline with value iteration function of MATLAB toolbox [15]. The resulting MDP policies are stored in *txt* format in the *mdp* folder of our *JavaScript* DASH player. This table is then read as the main adaptation rule as explained in Section III-B.

B. MDP-QL DASH Player

Recall that our MDP-QL based DASH player starts its learning process by selecting highly random quality levels. After around 2 hours driving in NICTA-UNSW route and 12 times streaming the Big Buck Bunny video clip, Boltzmann's temperature parameter (i.e., θ) became very small and close to zero (0.0001), and the Q-table values were updated more than 3500 times. Although to obtain the optimum policy via MDP-QL, we need to repeat the process until the Q-table values converge, we used the θ value as a threshold to stall the learning process and minimize the costs. However, we found that after θ drops to 0.0001, the generated policy becomes very close to the optimum policy.

Using *localStorage* function in our *JavaScript* code, we save both Q-table and θ after receiving every video chunk. These variables are then read and get updated after receiving the next video chunk. We repeat this process for every video chunk until the value of θ drops to 0.0001 which indicates that player has trained enough and is ready for real tests.

C. Video Statistics

To test our video players, we use the existing manifest files provided by DASHIF [5]. Table I presents the 10 available bitrates and resolutions of the *Big Buck Bunny* that is used in our experiments.

TABLE I: Available quality levels - video clip: *Big Buck Bunny*

Q	Kbps	Width	Height
1	230	224	128
2	331	284	160
3	477	368	208
4	688	448	252
5	991	592	332
6	1427	768	432
7	2056	992	560
8	2962	1280	720
9	5027	1600	900
10	6000	1920	1080

D. MDP Parameters

Recall that the tunable MDP parameters allow us to customize the streaming quality. In our models we use D to penalize the *state/action* pairs that cause deadline misses. In our previous study [6], we presented a wide range of results by tuning D in order to have fair comparison with different methods. In this study, we first run the *basic* DASH player. Then, we run multiple tests with different D values in our MDP-based players to find such a setting to achieve Average Quality (AQ) similar to the ones from *basic* DASH player. By fixing the AQ in all players, we can compare them based on their total number of quality changes (QC) and deadline misses (DM). We use ten times the quality level as fixed reward parameter, returning higher values for taking higher actions to encourage the Q-learning algorithm to maximize the AQ. Other MDP parameters [1] are set as following:

D=100, N= 10, M = 6, T = 2, n= 1, $\alpha = 0.9$, $\gamma = 0.9$, θ is initiated with 15 and ϵ = 0.005. Note that buffer length in this setting is $M \times T \times n = 12$ second. To have a fair comparison with DASHIF player, we set its buffer length to be 12 second as well. This can be done in *BufferExtension* class as discussed in subsection III-A.

[1] These parameters are explained in [6] and [7]

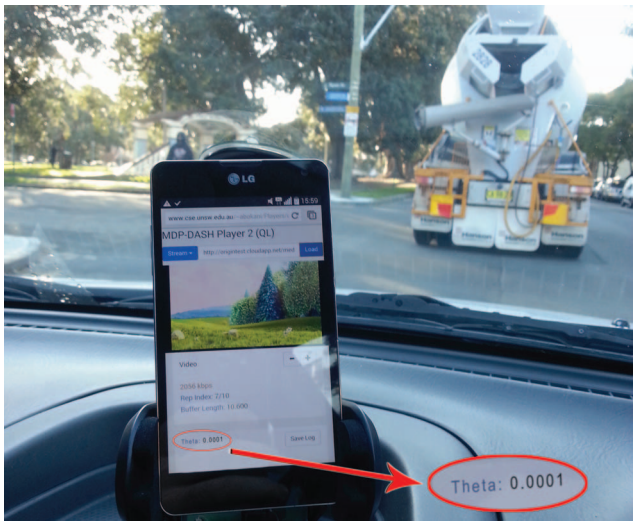


Fig. 4: Streaming Big Buck Bunny video clip on real-time basis while driving in NICTA-UNSW route

TABLE II: Driving Sessions

Driving Session	Date	Starting Time	Finishing Time
Data Collection	23/03/2015	11:32 AM	09:34 PM
	24/03/2015	10:00 AM	10:00 PM
	25/03/2015	10:42 AM	09:12 PM
	26/03/2015	10:29 AM	08:53 PM
Q-learning Process	08/04/2015	11:56 AM	02:30 PM
Testing	24/06/2015	10:30 AM	6:20 PM
	25/06/2015	11:15 AM	4:25 PM

E. Real-time Vehicular Streaming Tests

After preparing both MDP-VI and MDP-QL based players, we run our experiments as follows. Using an Android mobile phones, we played the Big Buck Bunny video clip on each player over real-world 3G network while driving in our NICTA-UNSW route (Fig. 2). The streaming logs were saved after finishing each playback session of our 9:56 length video clip. This process was repeated 10 times for all the three players on morning, afternoon, evening and night times to include peak and off-peak network conditions. In total, around 300 minutes streaming logs were recorded during 26 trips (Fig. 4) which will be used in our evaluations in the next section.

V. Results

In this section we present results from our driving tests. As discussed earlier, there are three metrics for video quality, picture resolution (AQ), number of quality changes (QC) and number of deadline misses (DM). After a few pilot driving tests with DASHIF player, we found that the AQ was approximately 5.5. We therefore tuned the parameters of the MDP players to achieve similar AQ, which leaves us with the remaining two metrics for comparison. We also observed the buffer level for these three players.

Recall that MDP-QL requires a training phase before it starts to produce its best performance. We have therefore completed some additional driving for the MDP-QL player before starting the testing phase. As shown in Table II, data collection, MDP-QL training, and the actual test drivings were separated from each other by more than a week, which allowed us to take into account the impact of any network variation during these gaps. Before we present the test results, we show the evolution of the MDP-QL player during its training phase.

TABLE III: Learning process with MDP-QL DASH player: quality of single streaming session in different moments of learning process

	Beginner	Trained
Average Quality (AQ)	5.34	5.51
# of Deadline Misses (DM)	43	0
# of Quality Changes (QC)	168	74
Available Buffer (AB)	4.92	9.97

Fig. 7 presents the buffer level (available buffered video) and the current quality selection at the beginning and end of the training phase. At the start, we have $\theta = 15$, which forces MDP-QL player to make very random decisions and learn from them. Fig. 5 shows the quality selections for 298 chunks starting from $\theta = 14.5$ (after watching 100 chunks). We see that due to highly randomness of quality selections, the number of quality changes is very high (168) while the buffer level is relatively low (4.92) and 43 deadline misses occurred. On the contrary, in Fig. 6, which shows performance from another 298 chunks after watching 3000 chunks ($\theta = 0.0001$), we see that the quality selections are more consistent around the average value of 5.51 with more available buffer and no deadline miss (see Table. III for actual statistics). Clearly the MDP-QL player was adequately trained after watching 3000 chunks, which took approximately 2 hours of download time with an average 3G bandwidth of 1802 kbps.

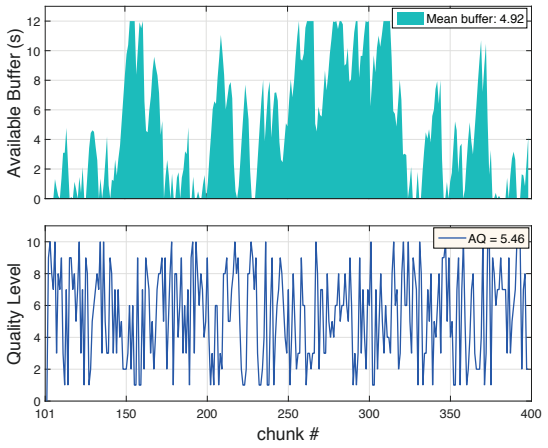
Next, we compare the performance of our MDP-based DASH players with DASHIF's player in Fig. 12, which shows average results from 10 repetitions of watching Big Buck Bunny. The buffer level, average quality, number of DMs and QCs are stored in the saved log files as discussed in Section III. As illustrated in Fig. 8, both MDP players buffered more video chunks compared to DASHIF. In particular, DASHIF had only 3.42 buffer available on average, which posed a great risk for deadline miss, while MDP-VI and MDP-QL recorded 8.1, and 9.14 available buffers, respectively. Total number of DM occurred with MDP-VI and MDP-QL DASH players were 4.1 and 3 respectively, which are 8x and 11x times less than the 34.2 DMs experienced with DASHIF player. Finally, number of quality changes for MDP players were also lower than those experienced with DASHIF.

VI. Conclusions

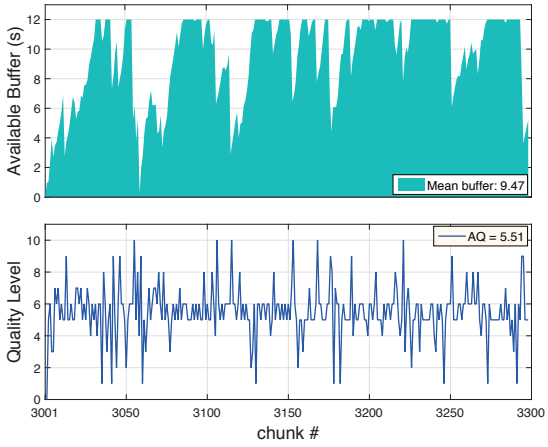
Although previous work had predicted the potential of MDP for improving streaming quality for DASH, its experimental validation had not been reported. In this paper, we implement MDP-based DASH player in Android platform and compare its performance against the default DASH player under real driving conditions. Our results show that use of MDP to adapt video qualities can reduce video freezing or buffering events significantly compared to the default DASH player while achieving comparable picture resolutions.

References

[1] Apple, "HTTP Live Streaming Overview," [Online accessed 01-March-2013], URL: https://developer.apple.com/streaming/.

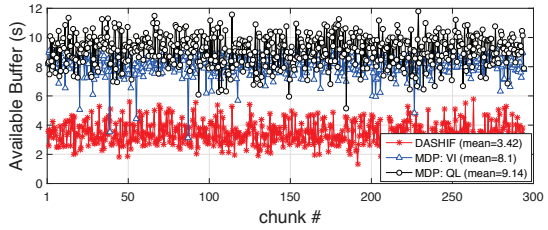


(a) Beginning of learning process: $\theta = 14.5$

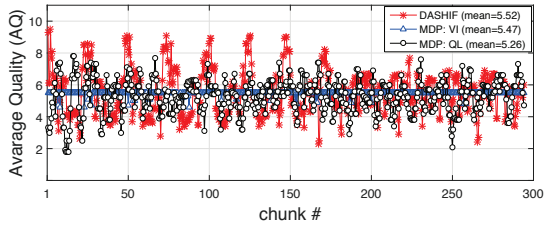


(b) After learning process: $\theta = 0.0001$

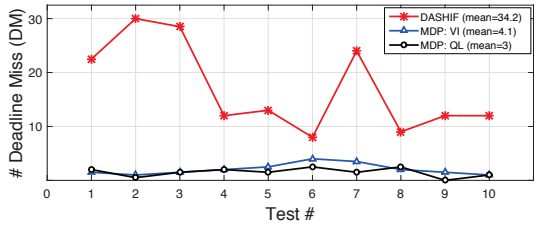
Fig. 5: Learning process with MDP-QL DASH player, Clip: Big Buck Bunny



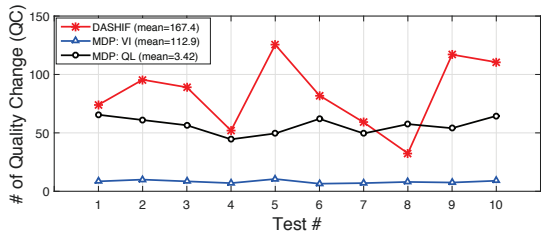
(a)



(b)



(c)



(d)

Fig. 6: Comparing basic and MDP-based DASH players: (a) Available Buffer, (b) Averagte Quality, (c) Deadline miss, (d) Quality change - Video clip: Big Buck Bunny

[2] A. Zambelli, "IIS smooth streaming technical overview," *Microsoft Corporation*, vol. 3, 2009.

[3] Adobe, "HTTP Dynamic Streaming on the Adobe Flash Platform," [Online accessed 04-March-2013], URL: http://www.adobe.com/au/products/hds-dynamic-streaming.html.

[4] T. Stockhammer, "Dynamic Adaptive Streaming Over HTTP: Standards and Design Principles," in *Proceedings of the second annual ACM conference on Multimedia systems (MMSys)*, USA, 23 Feb 2011.

[5] D. I. Forum, "JavaScript DASH Player," [Online accessed 01-March-2014], URL: https://github.com/Dash-Industry-Forum/dash.js/.

[6] A. Bokani, M. Hassan, and S. S. Kanhere, "HTTP-Based Adaptive Streaming for Mobile Clients using Markov Decision Process," in *Proceedings of the 20th Packet Video Workshop (PV)*, San Jose, USA, 12 Dec 2013.

[7] A. Bokani, M. Hassan, S. Kanhere, and X. Zhu, "Optimizing HTTP-Based Adaptive Streaming in Vehicular Environment using Markov Decision Process," *Multimedia, IEEE Transactions on*, vol. 17, no. 12, pp. 2297–2309, 2015.

[8] A. Bokani, "Location-Based Adaptation for DASH in Vehicular Environment," in *Proceedings of the 2014 CoNEXT on Student Workshop ACM '2014*, Sydney, Australia, 2-5 December 2014.

[9] J. Van Der Wal, *Stochastic Dynamic Programming*. Methematisch Centrum, Amsterdam, The Netherlands, 1980.

[10] C. J. C. H. Watkins, "Learning From Delayed Rewards." Ph.D. dissertation, University of Cambridge, 1989.

[11] G. Zhong and A. Bokani, "A Geo-Adaptive JavaScript DASH Player," in *Proceedings of the VideoNEXT '2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming ACM '2014*, Sydney, Australia, 2-5 December 2014.

[12] O. Signal, "Network Monitoring," [Online accessed 01-March-2015], URL: http://opensignal.com/rg/.

[13] Ookla, "Internet Speed Test," [Online accessed 01-March-2015], URL: http://www.netindex.com/.

[14] Parse, "Perfect Cloud to Power Your App on any Platform," [Online accessed 01-March-2015], URL: https://www.parse.com.

[15] Mathworks, "Markov Decision Process Toolbox," [Online accessed 20-November-2012], URL: http://www.mathworks.com.au/matlabcentral/fileexchange/25786-markov-decision-processes-mdp-toolbox.