Connection Science
Vol. 00, No. 00, Month 200x, 1–20

RESEARCH ARTICLE

Design and Optimization of a (FA)Q-Learning-based HTTP Adaptive Streaming Client

Maxim Claeys*[a], Steven Latré[b], Jeroen Famaey[a], Tingyao Wu[c], Werner Van Leekwijck[c] and Filip De Turck[a]

[a]*Department of Information Technology, Ghent University - iMinds, Gaston Crommenlaan 8/201, B-9050 Gent, Belgium*
[b]*Department of Mathematics and Computer Science, University of Antwerp - iMinds, Middelheimlaan 1, B-2020 Antwerpen, Belgium*
[c]*Alcatel Lucent Bell Labs, Copernicuslaan 50, B-2018 Antwerpen, Belgium*

(*Received 00 Month 200x; final version received 00 Month 200x*)

In recent years, HTTP Adaptive Streaming (HAS) is becoming the de-facto standard for adaptive video streaming services. A HAS video consists of multiple segments, encoded at multiple quality levels. State-of-the-art HAS clients employ deterministic heuristics to dynamically adapt the requested quality level based on the perceived network conditions. Current HAS client heuristics are however hardwired to fit specific network configurations, making them less flexible to fit a vast range of settings. In this article, a (Frequency Adjusted)Q-Learning HAS client is proposed. In contrast to existing heuristics, the proposed HAS client dynamically learns the optimal behaviour corresponding to the current network environment in order to optimize the Quality of Experience (QoE). Furthermore, the client has been optimized both in terms of global performance and convergence speed. Thorough evaluations show that the proposed client can outperform deterministic algorithms by 11% to 18% in terms of Mean Opinion Score (MOS) in a wide range of network configurations.

Keywords: HTTP Adaptive Streaming, reinforcement learning, agent systems, Quality of Experience

1. Introduction

Over the past decades, multimedia services have gained a lot of popularity. This growth is largely due to video streaming services. These services can generally be divided into Internet Protocol Television (IPTV), offered by a network provider and managed through resource reservation, and Over-The-Top (OTT) services, streamed over a network provider's network without his intervention (e.g. YouTube[1] and Netflix[2]). HTTP Adaptive Streaming (HAS) techniques are becoming the de-facto standard for OTT video streaming. Large industrial players such as Microsoft, Apple and Adobe have commercial implementations of the HAS concept available. These HTTP-based techniques split video content into small segments of typically 2s to 10s, encoded at multiple quality levels. This approach allows video clients to dynamically adapt the requested video quality to fit the perceived network state. Based on the perceived characteristics, such as delay and throughput, a quality selection heuristic is used at the client side to determine the

*Corresponding author. Email: maxim.claeys@intec.ugent.be
[1]http://www.youtube.com
[2]http://www.netflix.com

ISSN: 0954-0091 print/ISSN 1360-0494 online

DOI: 10.1080/09540090xxxxxxxxxxxx
http://www.informaworld.com

quality level to request for the next segment, in order to maximize the Quality of Experience (QoE).

HAS comes with important advantages. Not only is the video content delivered reliably over HTTP, HAS also allows seamless interaction through firewalls. On the downside, delivery over the best-effort Internet makes these techniques prone to network congestion and large bandwidth fluctuations due to cross traffic, which can be detrimental for the QoE, the quality as perceived by the end-users. HAS client behaviour is therefore a crucial factor for the streaming service to be beneficial and to ensure a sufficient level of QoE for the end user.

Current HAS client heuristics are however hardcoded to fit specific network configurations. This makes current approaches less flexible to deal with a vast range of network setups and corresponding bandwidth variations. Even though they are well suited for a specific network configuration, these deterministic approaches yield unsatisfactory results when the environment changes. This article proposes a Reinforcement Learning (RL) based HAS client, allowing dynamic adjustment of streaming behaviour to the perceived network state. RL is a machine learning technique, designed to operate in situations in which an agent only has limited knowledge about the environment, leading to a high degree of uncertainty concerning how the environment will react to the performed actions. However, interaction with the environment is the only way for the agent to learn. At each state in the environment, the agent perceives a numerical reward, providing feedback to the agent's actions. The agent's goal is to learn which action to take in a given state of the environment, in order to maximize the cumulative numerical reward (Kaelbling, Littman, & Moore, 1996). Mapping this RL principle to the HAS scenario, the agent learns which quality level to request in the perceived network state.

The contributions of this paper are three-fold. First, a Q-Learning-based HAS client has been designed. This approach, in contrast to traditional heuristics, allows the client to dynamically learn the best actions corresponding to the actual network environment. Second, a Frequency Adjusted Q-Learning (FAQ-Learning) approach is proposed to increase the client performance in strongly variable environments. Third, an estimation algorithm is presented to incorporate HAS domain knowledge into the initial Q-Tables in order to boost the client performance during the learning phase. All of the presented approaches are thoroughly evaluated using a network-based video streaming simulation framework. The simulation results allow comparison with the Microsoft ISS Smooth Streaming algorithm, of which the original source code is available.

The remainder of this article is structured as follows. First, the basic HAS principle is discussed in Section 2. Next, Section 3 gives an overview of related work, both on HAS and RL. Section 4 elaborates on the design of the proposed self-learning HAS client. Next to a general overview, this section presents the applied RL techniques, exploration policies and the constructed environmental state and reward function. Furthermore, in Section 5, the initial Q-Table estimation algorithm is proposed. The evaluations of the presented self-learning HAS client are described in Section 6. Finally, Section 7 presents some final conclusions.

2. HTTP Adaptive Streaming

HAS is the third generation of HTTP based streaming and is increasingly being used in OTT video delivery. Several large industrial players have commercial implementations of the HAS concept, including Microsoft ISS Smooth Stream-

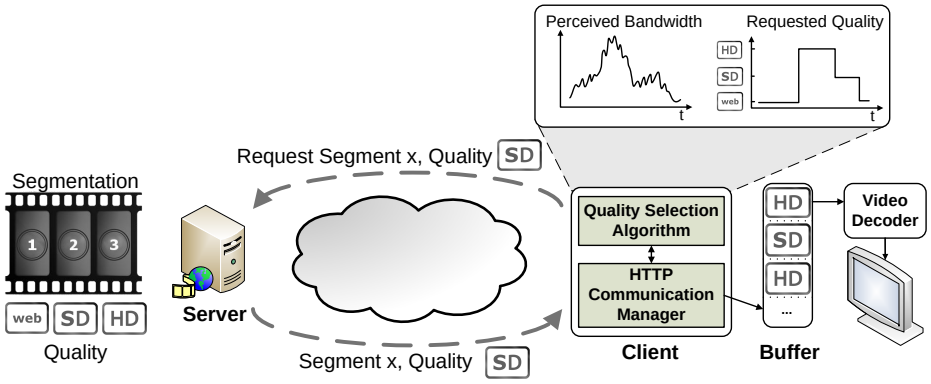


Figure 1. Schematic overview of the HTTP Adaptive Streaming (HAS) concept.

ing (MSS)[1], HTTP Live Streaming (HLS) by Apple[2] and Adobe's HTTP Dynamic Streaming[3]. In 2011, MPEG tried to find the common ground between the vast amount of commercial implementations by standardizing the interfaces and protocol data in Dynamic Adaptive Streaming over HTTP (DASH) (Stockhammer, 2011). The bitrate adaptation heuristics are, however, not standardized, and thus implementation specific.

Regardless of heuristic details, all of these implementations follow the general HAS concept, shown in Figure 1. In HAS, a video consists of multiple segments with a typical length of 2s to 10s, encoded at multiple quality levels. At the client side, a manifest file, containing information about the segments and quality levels, is used to link the different segments into a single video stream. Based on the information in the manifest file, the HAS client sequentially requests the next segment upon arrival of the previous segment. Based on the network state, perceived while downloading previous segments, a quality selection heuristic dynamically adapts the requested quality level in order to optimize the QoE. Each segment is downloaded in a progressive manner, while a buffer at the client side is used to take care of temporary anomalies such as a late arrival of a video segment. Finally, the video segments, stored in the buffer, are played back as a single continuous video stream. Current quality adaptation algorithms for HAS are deterministic and tailored to specific network configurations, hampering the ability to react to a vast range of highly dynamic network settings. On the contrary, this article proposes a self-learning HAS client to autonomously react to changing network conditions, as will be discussed in Section 4.

3. Related work

3.1. *HAS client algorithms*

As described in Section 2, multiple proprietary HAS algorithms are available. Akhshabi, Begen, and Dovrolis (2011) compare several commercial and open source HAS clients and identify their inefficiencies, such as excessive quality switching. Recently, several new client approaches have been described in literature. Liu, Bouazizi, and Gabbouj (2011) propose a client heuristic to handle parallel HTTP connections, based on the segment download time. By comparing the perceived

[1]http://www.iis.net/downloads/microsoft/smooth-streaming
[2]http://tools.ietf.org/html/draft-pantos-http-live-streaming-10
[3]http://www.adobe.com/products/hds-dynamic-streaming.html

segment download time with the expected segment download time, bandwidth fluctuations can be estimated appropriately. The opportunities of HAS in the domain of live streaming services are investigated by Lohmar, Einarsson, Frojdh, Gabin, and Kampmann (2011). The work focusses on the influence of client buffering on the end-to-end delay. Many recent research topics focus on the applicability of HAS in mobile environments by exploiting additional information. The heuristic described by Riiser, Vigmostad, Griwodz, and Halvorsen (2011) uses Global Positioning System (GPS) information to obtain more accurate information on the available bandwidth. Furthermore, Adzic, Kalva, and Furht (2011) have proposed content-aware heuristics. These approaches require metadata to be embedded in the video description. The additional consequences of quality selection in mobile environments have been shown by Trestian, Moldovan, Ormond, and Muntean (2012). The research shows that lowering the requested quality can significantly reduce energy consumption of Android devices. Jarnikov and Özçelebi (2010) discuss several guidelines on configuring robust HAS clients with regard to changing network conditions. The authors model the quality selection problem as a Markov Decision Process (MDP) which is solved offline. The performance is only guaranteed when the resulting strategy is applied in the same network environment as was modelled in the MDP. The same rationale holds for the dynamic programming approach, proposed by Xiang, Cai, and Pan (2012), where the bandwidth transition probabilities are needed to calculate the client policy offline.

In contrast to the above described approaches, we focus on an increased adaptivity and self-learning behaviour of the client heuristic through the design of a RL-based client approach. An initial approach to the application of a self-learning agent for HAS has been presented by Claeys et al. (2013). Even though the presented approach showed promising results, further experiments have shown that the performance in a variable networking environment was unsatisfactory. Therefore, the client has been thoroughly redesigned from the ground by reducing the environmental state space and reward definition. Furthermore, this article presents several techniques to further boost the performance and multiple network configurations have been evaluated. The use of a RL agent in HAS clients has also been proposed by Menkovski and Liotta (2013), applying the SARSA(λ) technique. Even though convergence of the learning agent is shown, a general evaluation of the client is infeasible since no comparison to existing approaches is provided.

3.2. *Learning in adaptive streaming*

Even though learning has not been applied frequently in the area of HAS, multiple RL-based adaptive streaming techniques have been proposed in the literature. Where our self-learning HAS approach is focused on the client side, existing RL-based adaptive streaming techniques target server or network side solutions to Quality of Service (QoS) provisioning for adaptive streaming systems. Fei, Wong, and Leung (2006) formulate call admission control and bandwidth adaptation for adaptive multimedia delivery in mobile communication networks as a Markov Decision Problem (MDP), which they solve using Q-Learning. RL is applied by Charvillat and Grigoras (2007) to create a dynamic adaptation agent, considering both user behaviour and context information. Furthermore, this generic approach is applied to solve a ubiquitous streaming problem. Artificial neural networks are used by McClary, Syrotiuk, and Lecuire (2008) to dynamically adapt the audio transmission rate in mobile ad hoc networks, considering available throughput, end-to-end delay and jitter.

3.3. *Learning in QoS/QoE optimization*

Reinforcement Learning (RL) has previously been successfully applied to various network management problems. Cao (2011) proposes an agent-based network fault diagnosis model in which the agent uses RL to improve its fault diagnosis performance. Their research shows this approach can outperform traditional fault diagnosis models. Bagnasco and Serrat (2009) propose the application of RL to dynamically adapt a hierarchical policy model to perform autonomous network management. They argue that an autonomic system must have a degree of flexibility to adapt to changes in goals or resources, which is hard to achieve by means of static policies. In the area of resource allocation, successfull applications of RL can be found. Vengerov (2005) presents a general framework for adaptive reconfiguration of distributed systems using a combination of RL and fuzzy rulebases. Dynamic resource allocation of entities sharing a set of resources is used as an example. On the other hand, Tesauro, Jong, Das, and Bennani (2007) propose a hybrid approach, gaining performance in the combination of RL and deterministic queuing models for resource allocation. In this hybrid system, RL is used to train offline on collected data, hereby avoiding possible performance loss during the online training phase. Furthermore, multiple approaches have been proposed, focussing on the resource allocation aspect in wireless mesh networks (Lee, Marconett, Ye, & Yoo, 2007; Niyato & Hossain, 2006). Another area of network management where RL has been applied previously is QoS routing. Especially in wireless sensor networks, the network topology may change frequently, yielding inherently imprecise state information, which impedes QoS routing. Ouferhat and Mellouk (2007) propose a Q-Learning based formalism to optimise QoS scheduling. Parakh and Jagannatham (2012) propose a decentralized bandwidth allocation for video streams in wireless systems, based on game theory. In this system, users are charged for bandwidth resources proportionally to the requested bit-rate. Mastronarde and van der Schaar (2011) apply RL to the problem of energy-efficient point-to-point transmission of delay-sensitive (multimedia)data over a wireless communication channel.

4. Reinforcement learning-based HAS client

4.1. *Approach*

As discussed in Section 2, current deterministic HAS clients only have limited abilities to react to a vast range of dynamic network settings. We propose the usage of a learning agent to enable the HAS client to adapt its behaviour by interacting with the network environment. In this way, the client will be able to react to network conditions that were not under consideration when designing the typical deterministic quality selection algorithms.

4.2. *Q-Learning*

A commonly used RL algorithm is Q-Learning (Sutton & Barto, 1998). Using Q-Learning, knowledge regarding both reward prospects and environmental state transitions are obtained through interaction with the environment. In Q-Learning, Q-values $Q(s, a)$ are used to measure the "Quality" of taking a specific action a in a certain state s, based on the perceived rewards. By applying eligibility traces (Sutton & Barto, 1998), current rewards are not only credited to taking the last action, but also to actions taken further in the past. Therefore, additional variables $e(s, a)$ are introduced for every state-action pair (s, a), indicating the

degree to which taking action a in state s is *eligible* for undergoing learning changes when a new reward is perceived.

Equations (1) and (2) respectively show how the eligibility traces and the Q-values are updated when action a is taken in state s, yielding a reward r and new state s'. In these equations, (s, a) is the state-action pair and $\alpha \in [0; 1]$ and $\gamma \in [0; 1]$ are the learning rate and the discount factor respectively. The parameter λ is referred to as the trace-decay parameter. I_{xy} denotes an identity indicator function, equal to 1 if $x = y$ and 0 otherwise.

$$e(x, y) = I_{xs} \cdot I_{ya} + \begin{cases} \lambda\gamma e(x, y) & : Q(s, a) = \max_b Q(s, b) \\ 0 & : \text{else} \end{cases} \tag{1}$$

$$Q(s, a) = Q(s, a) + \alpha e(s, a) \left[r + \gamma \max_b Q(s', b) - Q(s, a) \right] \tag{2}$$

The action to be performed in a specific state is selected based on the learned Q-values. The specific selection tactic depends on the used policy (see Section 4.4).

4.3. *Frequency Adjusted Q-Learning*

One problem that occurred using Q-Learning in preliminary simulations is the slow reaction to changes in the environment. When an action has a significantly lower Q-value than another action for a specific state, it has very low probability to be selected. When the environment changes, the new information on the quality of the action is only obtained very slowly because the action is unlikely to be selected and the Q-values are only adapted slowly, especially when using a small learning rate. To address this issue, we investigated a variant of the FAQ-Learning technique, proposed by Kaisers and Tuyls (2010) for multi-agent RL. In the proposed technique, the Q-values are updated as defined by Equation (3) where $P(s, a)$ is the probability of taking action a in state s.

$$Q(s, a) = Q(s, a) + \min\left(\frac{\alpha}{P(s, a)}, 1\right) e(s, a) \left[r + \gamma \max_b Q(s', b) - Q(s, a) \right] \tag{3}$$

Using this update rule, updates are percolated faster when an action has low selection probability. The cut-off at 1 is needed to avoid overflow of Q-values.

4.4. *Exploration policy*

One of the most challenging tasks in RL can be found in balancing between exploration and exploitation (Tokic & Palm, 2011). An often used approach to this tradeoff is the ϵ-greedy method (Watkins, 1989). Using this method, exploration comes down to random action selection and is performed with probability ϵ. The best action with respect to current estimates is thus exploited with probability $1-\epsilon$. Since the optimal configuration of the ϵ-parameter is very application dependent, rigorous tuning is required to obtain desirable results.

Another commonly used exporation method is Softmax (Sutton & Barto, 1998). In contrast to the ϵ-greedy method, with Softmax, action-selection is always performed in a probabilistic way. A Boltzmann distribution is used to rank the learned Q-values, based on which selection probabilities $P(s, a)$ are calculated using Equation (4) for every state-action pair (s, a). The positive parameter β called the

inverse temperature. As with the ϵ-greedy method, the parameter has to be tuned to balance the exploration rate for the specific application.

$$P(s,a) = \frac{e^{\beta Q(s,a)}}{\sum_b e^{\beta Q(s,b)}} \tag{4}$$

Tokic et. al. propose the Value-Difference Based Exploration with Softmax action selection (VDBE-Softmax) policy (Tokic, 2010; Tokic & Palm, 2011). With VDBE-Softmax, the ϵ-greedy and the Softmax policy are combined in a way that exploration is performed, using the Softmax probabilities defined in Equation (4), with probability ϵ. Greedy action selection is executed with probability $1-\epsilon$. Furthermore, a state-dependent exploration probability $\epsilon(s)$ is used instead of defining a global parameter. The $\epsilon(s)$ values are updated in every learning step, based on the difference in Q-values before and after that step, denoted as Δ. In this way, the agent is guided to be more explorative when knowledge about the environment is uncertain, indicated by large fluctuations in Q-values. When the agent's knowledge becomes certain however, the amount of exploration should be reduced. This behaviour is achieved by updating the $\epsilon(s)$ values according to Equation (5).

$$\epsilon(s) = \delta \frac{1 - e^{\frac{-|\Delta|}{\sigma}}}{1 + e^{\frac{-|\Delta|}{\sigma}}} + (1-\delta)\epsilon(s) \tag{5}$$

In this equation, the *inverse sensitivity* σ influences the exploration in a way that higher values of σ allow high levels of exploration only when the Q-value changes are large. Lower σ-values allow exploration even at smaller Q-value changes. The parameter $\delta \in [0;1]$ defines the relative weight of the selected action on the state-dependent exploration probability. A commonly used value for δ is the inverse of the number of actions since all actions should contribute equally to $\epsilon(s)$.

The proposed HAS client has been evaluated using both the Softmax and the VDBE-Softmax exploration policy.

4.5. *State & reward definition*

In our initial approach to a self-learning HAS client (Claeys et al., 2013), we proposed an environment model with six state variables, yielding over 2.5 million discrete states in the evaluated scenario. Using this large state definition, convergence issues arose, making the client unapplicable in situations with variable bandwidth. Based on this experience, the proposed learning agent uses a state definition, constructed of only two parameters: the current client buffer filling level and the available bandwidth perceived by the client. Both elements are continuous values and thus need to be discretized in order to apply the previously presented RL algorithms. The specific value ranges and number of discretization levels are shown in Table 1. In this table, B_{max} and T_{seg} respectively denote the maximum client buffer size and the segment duration in seconds. The number of quality levels and the highest possible throughput, e.g. the physical link capacity, are represented by N and BW_{max} respectively. In the considered scenario with a maximum client buffer size of 20s and a video stream with segments of 2s, available at 7 quality levels, this yields an environment model with 88 states.

Since the reward function is the fundamental guide for the RL agent to learn the desired policy, we want the reward function to be a measure for the QoE. For the construction of this reward function, three aspects of quality are considered, as identified by Mok, Chan, and Chang (2011): (i) the current video quality level, (ii)

Table 1. Proposed environmental state definition.

State element	Range	Levels
Buffer filling	$[0\ ;\ B_{max}]$sec	$\frac{B_{max}}{T_{seg}} + 1$
Bandwidth	$[0\ ;\ BW_{max}]$bps	$N + 1$

the switching in quality levels during the video playout and (iii) buffer starvations, leading to video freezes. The reward components for each of these aspects are constructed as shown in Equations (6), (7) and (8). For the quality level and switching components, a linear function is used. The bufferfilling component has been modelled using a linear function for a non-empty buffer as well, but a large punishment of -100 is given when the buffer is empty to avoid video freezes.

$$\mathrm{R}_{\text{quality}} = QL_i - N \tag{6}$$

$$\mathrm{R}_{\text{switches}} = -1.0 * |QL_i - QL_{i-1}| \tag{7}$$

$$\mathrm{R}_{\text{bufferfilling}} = \begin{cases} -100 & : B_i = 0 \\ B_i - B_{max} & : B_i > 0 \end{cases} \tag{8}$$

As shown in Equation (9), the total reward function is defined as the sum of these components.

$$\mathrm{R} = \mathrm{R}_{\text{quality}} + \mathrm{R}_{\text{switches}} + \mathrm{R}_{\text{bufferfilling}} \tag{9}$$

4.6. *Action definition*

The agent's action is to select which quality level to download for every video segment. As described in Section 2, a HAS client selects the quality level for the next segment upon arrival of the previous segment. The set of available actions corresponds to the available quality levels of the video sequence and is therefore static throughout the playout of a video sequence. The concrete number of actions depends on the video sequence. The quality levels of the video sequence used in this work are described in Section 6.1.

5. Initial Q-value estimation

5.1. *Rationale*

When learning starts, the agent has no knowledge about the environment and the quality of the actions. Therefore, the Q-values $Q(s, a)$ are initialized at a default value, regularly $Q(s, a) = 0$. Since the reward function, described in Section 4.5, produces negative values, unexplored state-action combinations will always be favorited by the exploration policy since they have the highest Q-values. Using strong negative default values would have the opposite effect, favoriting previously used actions.

For the client to be able to react to new, unseen states in an acceptable way, domain knowledge can be incorporated into the initial Q-Tables. However, care has

to be given to the magnitude of the Q-values in order not to fully restrict learning steps. When the initial Q-values magnitude is too high, the level of exploration is too limited, hampering the ability to find an acceptable solution. The goal is to design an initial Q-Table that allows to achieve higher performance when learning in unseen states, while reaching similar results as with standard Q-Learning in the converged state.

5.2. *Estimation algorithm*

Result: matrix *estimates* with estimated Q-values
initialize value estimate matrix *estimates*
foreach discrete buffer filling level buf **do**
  foreach discrete bandwidth level bw **do**
    //Estimate quality and buffer reward
    foreach quality level ql **do**
      $expectedDur \leftarrow \frac{bitrate[ql]*segmentDur}{averageBW[bw]}$
      $changeProb \leftarrow \frac{expectedDur}{300}$
      $totReward \leftarrow 0.0$
      foreach discrete bandwith level $nextBw$ **do**
        $transProb \leftarrow \begin{cases} 1.0 - changeProb : bw = nextBw \\ \frac{changeProb}{numBWs-1} \quad : else \end{cases}$
        $duration \leftarrow \frac{bitrate[ql]*segmentDur}{averageBW[nextBw]}$
        if $duration < segmentDur$ **then**
          $expectedChange \leftarrow \lfloor \frac{segmentDur}{duration} \rfloor$
        else
          $expectedChange \leftarrow -1 * \lceil \frac{duration}{segmentDur} \rceil$
        $newBuf \leftarrow buf + expectedChange$
        $reward \leftarrow (ql - maxQl) + ((newBuf * segmentDur) - maxBuf)$
        $totReward \leftarrow totReward + (transProb * reward)$
      $estimates[buf][bw][ql] \leftarrow totReward$
    //Estimate the average selected quality level
    $avgQl \leftarrow 0.0$
    foreach quality level ql **do**
      $prob \leftarrow$ probability of taking action ql in the specified state
      $avgQl \leftarrow avgQl + (prob * ql)$
    //Estimate switch reward
    foreach quality level ql **do**
      $reward \leftarrow estimates[buf][bw][ql]$
      $switchReward \leftarrow -1 * |ql - avgQl|$
      $estimates[buf][bw][ql] \leftarrow reward + switchReward$

Algorithm 1: Initial Q-value calculation algorithm.

Algorithm 1 estimates the average reward for every state-action pair, to be used as initial Q-values. The algorithm works as follows. First, for every state-action pair, the quality and buffer reward component are estimated. Based on the average bandwidth in the discrete bandwidth level bw and the average segment size for quality level ql, the estimated download duration $expectedDur$ is calculated. However, the actual bandwidth, available when downloading the next segment,

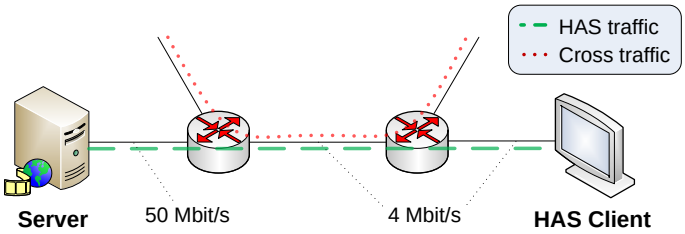


Figure 2. Overview of the simulated topology.

could vary. Therefore, we calculate the probability $changeProb$ that the available bandwidth level will change in the next $expectedDur$ seconds. To calculate this probability, we make the assumption that the available bandwidth remains stable for a uniform distributed amount of time between 1s and 300s. With probability $1 - changeProb$, the bandwidth level will stay the same, while any other bandwidth level has an equal probability $\frac{changeProb}{numBWs-1}$ to occur. For every bandwidth level $nextBw$, the average download duration $duration$ is calculated. Based on this duration and the segment duration $segmentDur$, the expected buffer change and new buffer filling level $newBuf$ can be calculated. Using this information, the quality and buffer filling reward components when the actual bandwidth is $nextBw$ are estimated. The influence of this value on the total reward is weighted by the probability $transProb$ that the available bandwidth level will be $nextBw$.

The next step in the algorithm is to estimate the average switch reward for every state-action pair. Therefore, the average quality level, selected in the specified state is calculated. In this calculation, the probability of taking an action in a certain state is given by the Softmax action-selection probability, defined in Equation 4. Knowing the average selected quality level $avgQl$, the average switch depth when selecting quality level ql can be estimated as $|ql - avgQl|$. Using this value, the estimated switch reward and the resulting total reward is calculated.

It is important to note that this algorithm is based on some assumptions and approximations. For example, a model for the available bandwidth is assumed when calculating the bandwidth shifting probability and averages are used when estimating the download duration. Therefore, the resulting values are only initial estimates and the self-learning client is needed to adapt the behaviour to the actual network environment. The complexity of the algorithm is linear in the number of discrete buffer filling levels and quality levels, and quadratic in the number of discrete bandwidth levels. Given the limited state-action space and the fact that the initial Q-values are only calculated once offline, the execution time is negligible.

6. Performance evaluation

6.1. *Experimental setup*

The experiments have been performed using the NS-3[1] based simulation framework described by Bouten et al. (2012). A network topology, shown in Figure 2, has been modelled, consisting of a single HAS client and server. This topology corresponds to a typical DSL access network scenario. On the last link on the path between the server and the client, a bandwidth capacity of 4Mbps is available for video delivery. At the client side, a maximum of 20s can be buffered.

[1]http://www.nsnam.org

Table 2. Proposed environmental state definition.

Quality level	Bitrate
1	300kbps
2	427kbps
3	608kbps
4	866kbps
5	1233kbps
6	1636kbps
7	2436kbps

Since the goal of the self-learning HAS client is to be able to deal with variable network environments, a highly variable bandwidth trace has been constructed by simulating cross traffic over a 3Mbps link and measuring the available throughput at the client side. The generated cross traffic is a sequence of bandwidth bursts, normally distributed between 0kbps and 2640kbps with a granularity of 264kbps. Each burst persisted for a uniformly distributed amount of time ranging from 1s to 300s. Using the resulting bandwidth trace not only yields high variability within an episode, but also across the episodes.

On this topology, the *Big Buck Bunny* video trace was streamed. A single episode of the video trace consists of 299 segments, each with a fixed length of 2s. Each segment has been encoded at 7 different quality levels, with bitrates ranging from 300kbps to 2436kbps, as shown in Table 2. To ensure the learning agent has time to converge, 400 episodes of the video trace are simulated.

The traditional Microsoft ISS Smooth Streaming (MSS) algorithm[1] is used to compare the behaviour of the learning client to current deterministic HAS algorithms. Using MSS, three buffer thresholds should be tuned to configure the client behaviour. For the panic, lower and upper buffer thresholds, the values of 25%, 40% and 80%, empirically determined by Famaey et al. (2013), have been used in our experiments.

To be able to draw meaningful conclusions when comparing the performance of the different clients, paired t-tests have been performed. Using paired t-tests, the significance of the difference between two approaches, applied in the same environment, can be shown. Furthermore, the comparison graphs in Section 6.3 contain error bars, visualizing the standard deviation of the plotted averages.

6.2. *Evaluation metrics*

The reward function, described in Section 4.5, has been constructed to be a measure of the quality of a decision on a single segment quality level. To evaluate the different approaches however, a measure of the total video playout quality has to be used. The QoE can only be estimated, either by subjective evaluation by a test panel or using an objective model of the users' perception. QoE of HAS video is still an active research topic and only a limited number of objective metrics are available. De Vriendt, De Vleeschauwer, and Robinson (2013) define the QoE of HAS video to be dependent on the average segment quality and the standard deviation of the segment quality. The parameters of the proposed *quality level model* were tuned based on the results of a small subjective test.

Next to the average quality level and the switching behaviour, video freezes are also considered to heavily impact the QoE of video delivery. However, video freezes are not considered in the model proposed by De Vriendt et al. (2013). The

[1]Original source code available from:
`https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming`

Table 3. Overview of evaluated parameter configurations.

	Parameter	Evaluated values
α	Learning rate	0.1 , 0.3 , 0.5 , 0.7 , 0.9
γ	Discount factor	0.1 , 0.3 , 0.5 , 0.7 , 0.9
λ	Eligibility trace-decay	0.1 , 0.5 , 0.6 , 0.7 , 0.9
β	Softmax inverse temperature	0.1 , 0.5 , 1.0 , 5.0

influence of video freezes depends both on the number and the average length of freezes (Mok et al., 2011). The calculation, proposed by Mok et al. (2011), uses only three discrete levels of freeze frequency and length. Based on an interpolation of these levels, a continuous function has been constructed to measure the impact of video freezes on QoE. The resulting function is shown in Equation (10), where F_{freq} and FT_{avg} represent the freeze frequency and the average freeze time respectively. Given that this function evaluates to 0 when no freezing occurs, the Mean Opinion Score (MOS) calculation proposed by De Vriendt et al. (2013) remains valid in a scenario without freezes.

$$\phi = \frac{7}{8} * \max\left(\frac{\ln(F_{freq})}{6} + 1, 0\right) + \frac{1}{8} * \left(\frac{\min(FT_{avg}, 15)}{15}\right) \tag{10}$$

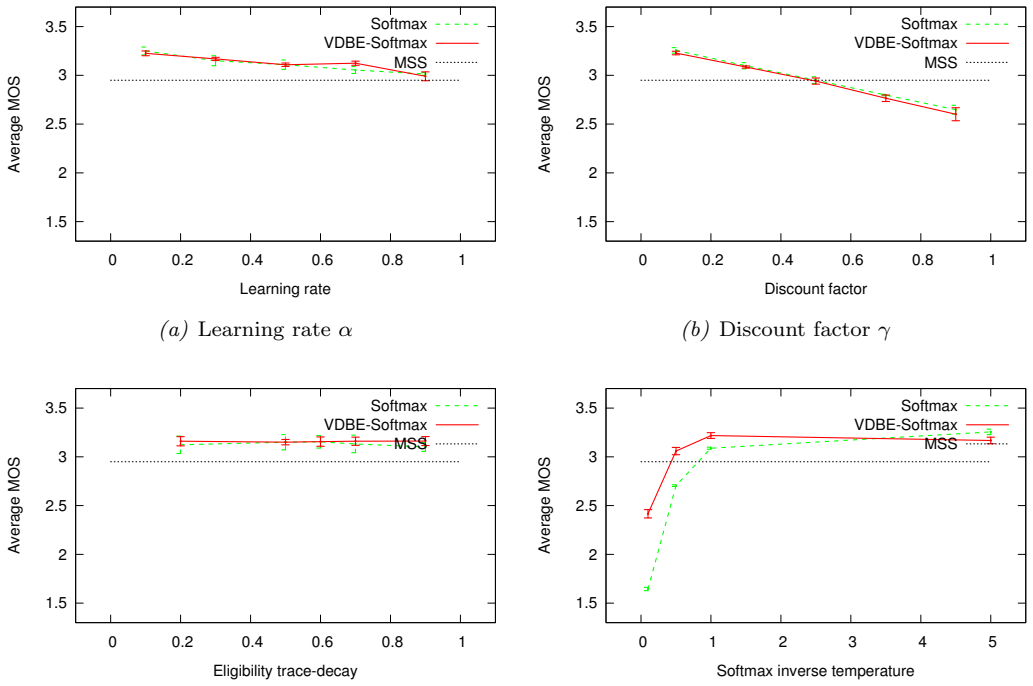
Combining the quality level, switching and video freezing aspects, the estimated MOS for the playout of a HAS video, consisting of K segments, playing quality level QL_k for segment k, can be calculated using Equation (11). In this equation, the average played quality level and its standard deviation are respectively represented by $\mu = \frac{\sum_{k=1}^{K} QL_k}{K}$ and $\sigma = \sqrt{\frac{\sum_{k=1}^{K} (QL_k - \mu)^2}{K}}$. One can verify that the theoretical range of this metric in a scenario with seven quality levels $[1; 7]$ is $[0.00; 5.84]$. During the simulations however, a practical metric range $[0.00; 5.06]$ was observed, which corresponds to the typical levels of a MOS.

$$\text{MOS}_{est} = \max\left(0.81 * \mu - 0.95 * \sigma - 4.95 * \phi + 0.17, 0\right) \tag{11}$$

6.3. *Results discussion*

6.3.1. Parameter analysis

As described in Section 4, both the Q-Learning algorithm and the exploration policies contain multiple parameters that can be tuned to optimize the behaviour. Given the continuous nature of the parameters and the mutual influence between them, it is unfeasible to evaluate all configurations to find the optimum. Therefore, a subset of 500 configurations has been evaluated for both the Softmax and the VDBE-Softmax policy using the Q-Learning algorithm with default initial Q-values. Based on preliminary experiments, the VDBE-Softmax inverse sensitivity parameter has been fixed to $\sigma = 1.0$. Using these configurations, the influence of every parameter can be analysed and an acceptable configuration can be selected. An overview of the evaluated parameter values can be found in Table 3. For the learning rate and discount factor, an evenly spaced selection of the value range was taken. The evaluated values for the eligibility trace-decay were centered around 0.6, empirically found to be a good performing configuration. For the selection of Softmax inverse temperature values, preliminary experiments have shown that the influence of the parameter fades out for values above 1.0.



(a) Learning rate α

(b) Discount factor γ

(c) Eligibility trace-decay λ

(d) Softmax inverse temperature β

Figure 3. Analysis of parameter influence.

To consider the converged state, for every configuration, the MOS has been calculated over the last 50 of 400 episodes. For each parameter, the average MOS of the best 5 configurations for every evaluated value of that parameter has been calculated. The results are shown in Figure 3. It is clear that each of the parameters has similar influence for both exploration policies. While a clear trend is shown for the learning rate, discount factor and Softmax inverse temperature, the system is rather insensitive to the eligibility trace-decay value. This behaviour can be explained by the strong preference of the system to low discount factors, strongly accelerating the decay, as defined by Equation (1). Since the VDBE-Softmax approach only applies the Softmax formula when exploring, the VDBE-Softmax approach is less sensitive to the value of the Softmax inverse temperature β. Finally, a wide range of parameters is shown to outperform the deterministic MSS algorithm.

Based on the analysis, the best configuration is determined to be $\alpha = 0.1$, $\gamma = 0.1$, $\lambda = 0.6$ and $\beta = 5.0$ using the Softmax exploration policy. Figure 4 shows the relative performance of the self-learning HAS client using this configuration compared to the traditional MSS client on the left axis. A moving average of the metric values of the last 50 episodes is presented in order to observe the general trend over the variable bandwidth episodes. The figure shows that after about 100 learning episodes, the client is able to achieve the same level of performance as the MSS client. The increasing trend stabilizes after about 200 episodes. The convergence of the learning agent can also be seen in the flattening out of the Q-value changes, plotted on the right axis. In the converged state, the self-learning HAS client is able to outperform the traditional MSS client with on average 10.31% in terms of average MOS in the last 50 episodes in a highly dynamic bandwidth environment. The performance increase is statistically significant with significance level 0.05 (two-tail paired t-test: $t = 8.5425$, $t_c = 2.0096$). With respect to the individual MOS factors, the MSS client is outperformed by 0.85%, 19.54% and 11.76% in terms of average quality level, average quality standard deviation and

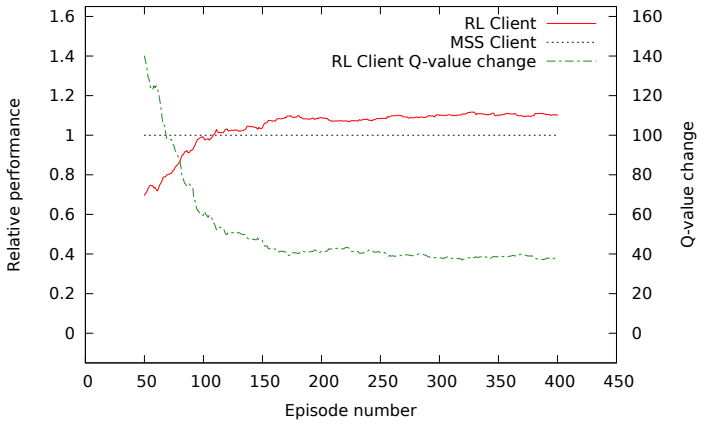


Figure 4. Convergence of the self-learning client performance, relative to the traditional MSS client.

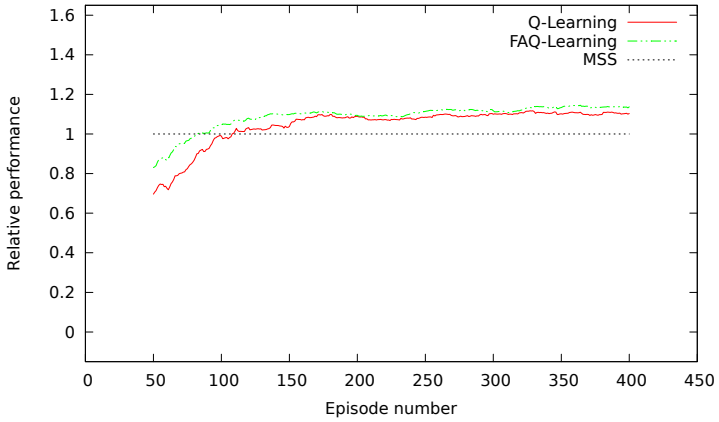


Figure 5. Relative performance of the FAQ-Learning and default Q-Learning approach compared to the traditional MSS client.

total freeze time respectively.

6.3.2. Frequency Adjusted Q-Learning

In Section 4.3, we argued that FAQ-Learning could possibly increase the performance of standard Q-Learning in strongly variable environments. To allow fair comparison between the performance of the FAQ-Learning and the standard Q-Learning client, both techniques have been applied in a highly dynamic bandwidth environment, using the parameter configuration selected in the previous section. In these simulations, again default initial Q-values have been used.

Figure 5 shows the relative performance of both clients compared to the traditional MSS client. The proposed FAQ-Learning approach clearly outperforms default Q-Learning, both in terms of convergence speed and absolute values. The performance increase is largely due to the smaller amount of freeze time. In Table 4, the performance of the FAQ-Learning client in the last 50 episodes is compared to the standard Q-Learning and MSS client in terms of average MOS and total freeze time. Using the proposed FAQ-Learning technique, the self-learning HAS client is able to outperform the traditional MSS client by 13.69% in terms of average MOS. This performance increase is statistically significant with significance level 0.05 (two-tail paired t-test: $t = 11.7688$, $t_c = 2.0096$). For the average quality level, average quality standard deviation and total freeze time, the achieved gain amounts 0.23%, 26.41% and 66.60% respectively.

Table 4. Performance comparison of the MSS, Q-Learning and FAQ-Learning client in terms of MOS and freeze time.

Client	Average MOS	σ_{MOS}	MOS Change*	Total Freeze time	Freeze time Change*
MSS	2.94986	0.65923	–	13.975s	–
Q-Learning	3.25403	0.63482	+10.31%	12.332s	-11.75%
FAQ-Learning	3.35369	0.59948	+13.69%	4.667s	-66.60%

*Compared to the traditional MSS client.

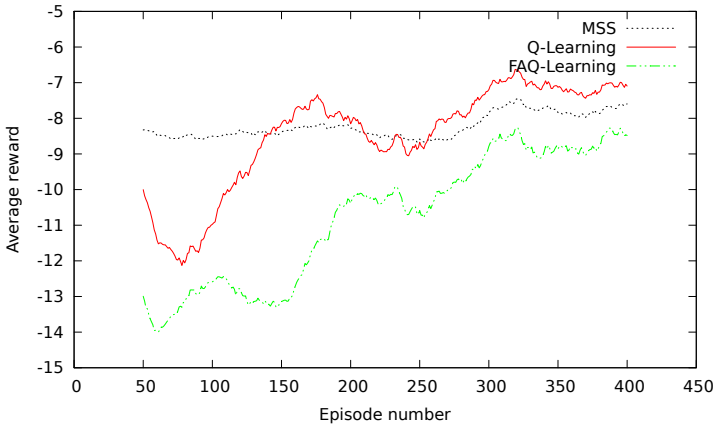


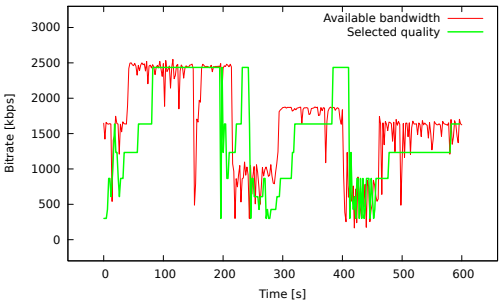
Figure 6. Reward performance of the FAQ-Learning and default Q-Learning approach compared to the traditional MSS client.

Compared to the approach using standard Q-Learning, a statistically significant (two-tail paired t-test: $t = 3.6382$, $t_c = 2.0096$) average MOS increase of 3.06% is obtained when applying FAQ-Learning. Despite of the performance gain in terms of QoE, the learning behaviour in terms of reward values is inferior to the approach using standard Q-Learning. As shown in Figure 6, lower reward values are obtained when applying FAQ-Learning, compared to the default Q-Learning approach. The explanation for these conflicting results can be found by analyzing the resulting quality selection behaviour of the three clients.

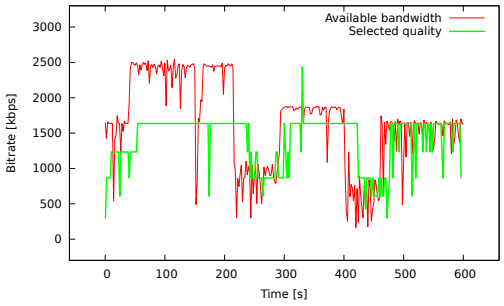
The quality selection behaviour of the MSS, Q-Learning and FAQ-Learning clients in episode 375 is illustrated in Figure 7. The resulting average reward component values for this episode are shown in Table 5. Both Figure 7 and Table 5 show that even though the MSS client is able to reach higher quality levels at some points, the Q-Learning client achieves overall higher average quality, lower standard deviation of quality level and higher average buffer filling. The Q-Learning client thus results in more stable behaviour. Moreover, it can be seen that using the FAQ-Learning client further increases the average quality level and decreases the quality level standard deviation. However, this comes at the cost of a lower buffer filling level, resulting in an overall lower reward value. Since buffer filling level is not directly influencing the QoE, the resulting MOS is not affected as long as the buffer is not fully depleted. As previously shown in Table 4, the lower buffer filling level does not introduce additional freezing time. Since the MOS is the aspect we aim to optimize in this use-case, the FAQ-Learning approach is preferred over the Q-Learning approach, despite of the lower average reward.

6.3.3. Initial Q-value estimation

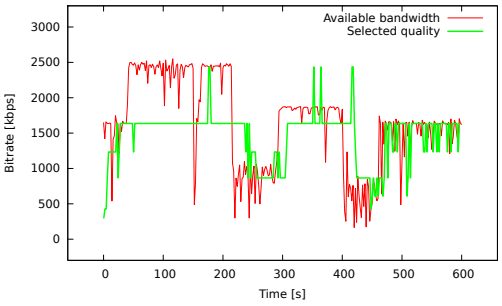
In Section 5, we proposed an algorithm to incorporate HAS domain knowledge into the initial Q-Table. Using this Q-Table, we target to drasticly improve the client performance in the learning phase while maintaining a similar performance



(a) Microsoft ISS Smooth Streaming



(b) Q-Learning



(c) FAQ-Learning

Figure 7. Behaviour comparison of the MSS, Q-Learning and FAQ-Learning HAS clients in episode 375 of the variable bandwidth scenario.

Table 5. Reward components of the MSS, Q-Learning and FAQ-Learning clients in episode 375 of the variable bandwidth scenario.

Client	**Reward component**			
	Quality	**Switches**	**Bufferfilling**	**Total**
MSS	-1.826	-0.365	-5.438	-7.629
Q-Learning	-1.632	-0.318	-4.294	-6.244
FAQ-Learning	-1.512	-0.278	-6.916	-7.950

level as with default Q-Tables in the converged state. The use of pre-calculated Q-Tables has been evaluated on four bandwidth traces with different levels of variability, using the Q-Learning technique.

- **Fixed:** throughout the entire simulation, a fixed bandwidth level of 2Mbps is maintained.
- **Sinus:** the bandwidth level is modelled by a sine function with a period of 600s and a codomain of [1Mbps;2Mbps].
- **Stepfunction:** every 20s, the bandwidth level switches between 1Mbps and 2Mbps.
- **Variable:** the highly variable bandwidth trace, as described in Section 6.1.

Figure 8 shows the performance of both approaches and the MSS client in the learning and converged state for each of the four bandwidth configurations. As in the rest of this article, we refer to the converged state as the last 50 episodes of the simulation of 400 episodes. The learning phase is defined as the first 50 episodes. In each of the bandwidth configurations, the self-learning client is shown to benefit from using the calculated Q-Tables in the learning phase. The statistical significance of the average MOS changes is presented in Table 6. The slight performance decrease in the converged state is caused by the reduced learning possibilities, introduced by the domain knowledge. However, the converged results are comparable to

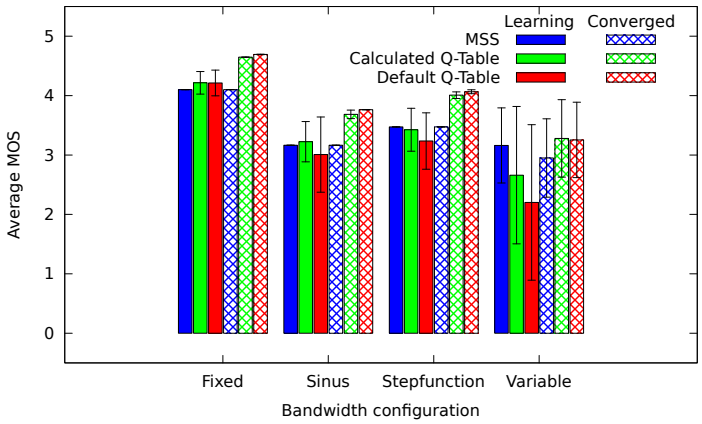


Figure 8. Performance comparison of the traditional MSS client and the self-learning client using default and calculated initial Q-Tables in the learning and converged phase.

Table 6. Statistical significance of average MOS differences using calculated initial Q-Tables. Significance results are obtained by two-tail paired t-testing with significance level 0.05.

Bandwidth Configuration	Phase	Default Q-Table	Calculated Q-Table	T-test Result	Statistically Significant*
Fixed	Learning[a]	4.21272	4.21582	0.1362	
	Converged[b]	4.69312	4.64692	83.4822	✓
Sinus	Learning[a]	3.00777	3.2242	2.8279	✓
	Converged[b]	3.76258	3.68460	7.6924	✓
Stepfunction	Learning[a]	3.23553	3.42564	2.8127	✓
	Converged[b]	4.06532	4.00774	6.3059	✓
Variable	Learning[a]	2.20202	2.66074	4.7204	✓
	Converged[b]	3.25403	3.27974	0.9528	

*Critical t-value: $t_c = 2.0096$.

[a]First 50 of 400 episodes.

[b]Last 50 of 400 episodes.

the performance when using default Q-Tables. Furthermore, the figure shows that the self-learning client, once converged, is able to outperform the deterministic MSS client on average by 11.18% (for the variable bandwidth configuration) to 18.89% (for the sinus bandwidth configuration) in terms of average MOS, depending on the bandwidth configuration. For the variable network configuration, the average quality level, average quality standard deviation and total freeze time have gained 1.52%, 20.38% and 5.45% respectively.

Besides from increasing the performance in terms of average MOS, compared to default initial Q-Tables, incorporating domain-knowledge in the initial Q-Tables strongly decreases the total freeze time in the learning phase. A comparison in terms of both MOS and total freeze time is given in Table 7 for the variable bandwidth configuration. The table shows that using pre-calculated initial Q-Tables compared to default values strongly boosts the client performance in the learning phase while reaching a similar performance level when converged. Even though an additional freeze time of about 900ms is introduced in the coverged state, the overall MOS, incorporating the freezes, is not affected. The increased freeze time is compensated by higher average quality level and lower quality standard deviation.

6.3.4. Results summary

Table 8 summarizes the results of the self-learning client approaches, compared to the traditional Microsoft ISS Smooth Streaming (MSS) client, in terms of both the average MOS and the individual quality components. It is shown that a Q-Learning

Table 7. Performance comparison of the Q-Learning-based client using default and calculated initial Q-Tables in terms of average MOS and total freeze time for the variable bandwidth configuration.

Phase	Initial Q-Table	Average MOS	σ_{MOS}	MOS Change*	Total Freeze time	Freeze time Change*
Learning[a]	Default	2.20202	1.30955	–	526.075s	–
	Calculated	2.66074	1.15649	+20.83%	252.460s	-52.01%
Converged[b]	Default	3.25403	0.63482	–	12.332s	–
	Calculated	3.27974	0.65199	+0.79%	13.214s	+7.15%

*Compared to the client using default Q-Tables.

[a]First 50 of 400 episodes.

[b]Last 50 of 400 episodes.

Table 8. Performance summary of the self-learning approaches in the variable bandwidth configuration, compared to the traditional MSS client, in terms of the quality components.

Technique	Initial Q-Table	MOS Change*	Quality Change*	Switching Change*	Freeze time Change*
Q-Learning	Default	+10.31%	+0.85%	-19.54%	-11.76%
FAQ-Learning	Default	+13.69%	+0.23%	-26.41%	-66.60%
Q-Learning	Calculated	+11.18%	+1.52%	-20.38%	-5.45%

*Compared to the traditional MSS client.

based HAS client outperforms the deterministic MSS client for each of the quality aspects. Using the proposed Frequency Adjusted Q-Learning (FAQ-Learning) technique, further improvement is obtained. Furthermore, the use of pre-calculated initial Q-Tables strongly boosts the performance in the learning phase and reaches similar results as with default Q-Tables in the converged state.

7. Conclusions

In this article, we designed a Reinforcement Learning (RL)-based HTTP Adaptive Streaming (HAS) client, dynamically adjusting its behaviour to the perceived networking environment. We presented an extended parameter analysis to fine-tune the client configuration to operate in a dynamic network environment. Next, we proposed using a Frequency Adjusted Q-Learning (FAQ-Learning) approach to strongly increase the client performance in variable environments. Furthermore, we presented an estimation algorithm to incorporate domain knowledge into the initial Q-Tables. Using these estimations, we were able to drasticly improve the clients performance during its learning phase, both in terms of average Mean Opinion Score (MOS) and total freeze time. The resulting self-learning HAS client is shown to outperform the deterministic traditional Microsoft ISS Smooth Streaming (MSS) client in terms of average MOS by 11% to 18% in all of the evaluated bandwidth scenarios with different degrees over variability, while increasing the performance for each of the identified MOS components.

Acknowledgments

M. Claeys is funded by grant of the Agency for Innovation by Science and Technology in Flanders (IWT). The research was performed partially within the ICON MISTRAL project (under grant agreement no. 10838). This work was partly funded by Flamingo, a Network of Excellence project (318488) supported by the European Commission under its Seventh Framework Programme. The Alcatel-Lucent research was performed partially within IWT project 110112.

References

Adzic, V., Kalva, H., & Furht, B. (2011). Optimized adaptive HTTP streaming for mobile devices. *Applications of Digital Image Processing XXXIV*, 81350T.

Akhshabi, S., Begen, A. C., & Dovrolis, C. (2011). An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proceedings of the second annual ACM conference on Multimedia systems* (pp. 157–168).

Bagnasco, R., & Serrat, J. (2009). Multi-agent Reinforcement Learning in Network Management. In *Scalability of Networks and Services* (Vol. 5637, pp. 199–202). Springer Berlin Heidelberg.

Bouten, N., Famaey, J., Latré, S., Huysegems, R., De Vleeschauwer, B., Van Leekwijck, W., & De Turck, F. (2012). QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming. In *2012 8th International Conference on Network and Service Management (CNSM)* (pp. 336–342).

Cao, J. (2011, June). Using reinforcement learning for agent-based network fault diagnosis system. In *2011 IEEE International Conference on Information and Automation (ICIA)* (pp. 750–754).

Charvillat, V., & Grigoras, R. (2007). Reinforcement learning for dynamic multimedia adaptation. *Journal of Network and Computer Applications*, *30*(3), 1034 - 1058.

Claeys, M., Latré, S., Famaey, J., Wu, T., Van Leekwijck, W., & De Turck, F. (2013). Design of a Q-Learning-based Client Quality Selection Algorithm for HTTP Adaptive Video Streaming. In *2013 Workshop on Adaptive and Learning Agents (ALA)*.

De Vriendt, J., De Vleeschauwer, D., & Robinson, D. (2013). Model for estimating QoE of Video delivered using HTTP Adaptive Streaming. In *1st IFIP/IEEE Workshop on QoE Centric Management, QCMAN 2013*.

Famaey, J., Latré, S., Bouten, N., Van de Meerssche, W., De Vleeschauwer, B., Van Leekwijck, W., & De Turck, F. (2013). On the Merits of SVC-Based HTTP Adaptive Streaming. In *2013 12th IFIP/IEEE International Symposium on Integrated Network Management (IM)*.

Fei, Y., Wong, V. W. S., & Leung, V. C. M. (2006, February). Efficient QoS provisioning for adaptive multimedia in mobile communication networks by reinforcement learning. *Mobile Networks and Applications*, *11*(1), 101–110.

Jarnikov, D., & Özçelebi, T. (2010, July). Client intelligence for adaptive streaming solutions. In *2010 IEEE International Conference on Multimedia and Expo (ICME)* (pp. 1499–1504).

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237-285.

Kaisers, M., & Tuyls, K. (2010). Frequency adjusted multi-agent Q-learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1* (pp. 309–316).

Lee, M., Marconett, D., Ye, X., & Yoo, S. (2007). Cognitive Network Management with Reinforcement Learning for Wireless Mesh Networks. In *IP Operations and Management* (Vol. 4786, pp. 168–179). Springer Berlin Heidelberg.

Liu, C., Bouazizi, I., & Gabbouj, M. (2011, August). Parallel Adaptive HTTP Media Streaming. In *Proceedings of the 20th International Conference on Computer Communications and Networks (ICCCN)* (pp. 1–6).

Lohmar, T., Einarsson, T., Frojdh, P., Gabin, F., & Kampmann, M. (2011, June). Dynamic adaptive HTTP streaming of live content. In *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (pp. 1–8).

Mastronarde, N., & van der Schaar, M. (2011). Fast Reinforcement Learning for Energy-Efficient Wireless Communication. *IEEE Transactions on Signal Processing*, *59*(12), 6262-6266.

McClary, D. W., Syrotiuk, V. R., & Lecuire, V. (2008). Adaptive audio streaming in mobile ad hoc networks using neural networks. *Ad Hoc Networks*, *6*(4), 524 - 538.

Menkovski, V., & Liotta, A. (2013). Intelligent control for adaptive video streaming. In *2013 IEEE International Conference on Consumer Electronics (ICCE)* (p. 127-128).

Mok, R., Chan, E., & Chang, R. (2011, May). Measuring the Quality of Experience of HTTP video streaming. In *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (pp. 485–492).

Niyato, D., & Hossain, E. (2006, June). A Radio Resource Management Framework for IEEE 802.16-Based OFDM/TDD Wireless Mesh Networks. In *IEEE International Conference on Communications, 2006. ICC '06.* (Vol. 9, pp. 3911–3916).

Ouferhat, N., & Mellouk, A. (2007, May). A QoS Scheduler Packets for Wireless Sensor Networks. In *IEEE/ACS International Conference on Computer Systems and Applications, 2007. AICCSA '07.* (pp. 211–216).

Parakh, S., & Jagannatham, A. (2012). Game theory based dynamic bit-rate adaptation for H.264 scalable video transmission in 4G wireless systems. In *2012 International Conference on Signal Processing and Communications (SPCOM)* (p. 1-5).

Riiser, H., Vigmostad, P., Griwodz, C., & Halvorsen, P. (2011, July). Bitrate and video quality planning for mobile streaming scenarios using a GPS-based bandwidth lookup service. In *2011 IEEE International Conference on Multimedia and Expo (ICME)* (pp. 1–6).

Stockhammer, T. (2011). Dynamic adaptive streaming over HTTP: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems* (pp. 133–144).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press. Hardcover.

Tesauro, G., Jong, N. K., Das, R., & Bennani, M. N. (2007, September). On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, *10*(3), 287–299.

Tokic, M. (2010). Adaptive ϵ-greedy exploration in reinforcement learning based on value differences. In *Proceedings of the 33rd annual German conference on Advances in artificial intelligence* (pp. 203–210). Springer-Verlag.

Tokic, M., & Palm, G. (2011). Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Proceedings of the 34th Annual German conference on Advances in artificial intelligence* (pp. 335–346). Springer-Verlag.

Trestian, R., Moldovan, A.-N., Ormond, O., & Muntean, G.-M. (2012, April). Energy consumption analysis of video streaming to Android mobile devices. In *Network Operations and Management Symposium (NOMS), 2012 IEEE* (pp. 444–452).

Vengerov, D. (2005). *A Reinforcement Learning Approach to Dynamic Resource Allocation* (Tech. Rep.). Sun Microsystems Laboratories.

Watkins, C. (1989). *Learning from Delayed Rewards* (Unpublished doctoral dissertation). University of Cambridge, England.

Xiang, S., Cai, L., & Pan, J. (2012). Adaptive scalable video streaming in wireless networks. In *Proceedings of the 3rd Multimedia Systems Conference.*