Cognitive Streaming on Android Devices

Maria Torres Vega*, Decebal Constantin Mocanu*, Rosario Barresi†, Giancarlo Fortino†, Antonio Liotta*
*Department of Electrical Engineering
Eindhoven University of Technology
Email: m.torres.vega, d.c.mocanu, a.liotta@tue.nl
†Dipartimento di Informatica, Elettronica e Sistemistica (DEIS)
University of Calabria
Email: r.barresi89@gmail.com, g.fortino@unical.it

***Abstract*—As the number of mobile devices increases, so do the complexity of wireless networks and the user's requirements. This tendency makes necessary for Multimedia Services to take the needed actions to adapt to the upcoming technology. A prominent example of this type of services is HTTP Adaptive Video Streaming Applications. In this research, we have studied how the latest HTTP Adaptive Streaming techniques, mainly developed for standard computers, could be adapted and used in mobile wireless devices. Furthermore, inspired by these solutions, which usually make use of Reinforcement Learning (RL) algorithms to find the suitable streaming rate, we have conceived a novel smart video player client in Java for Android platform using the Dynamic Adaptive Streaming over HTTP (DASH) protocol. We have assessed the performance of our proposed solution in a self-developed wireless test-bed under different network conditions. Thus, we have seen that by including in the reward function contributions regarding the download speed of the video segments, especially needed due to the fluctuating nature of the wireless networks, and the segments already buffered, improves drastically the overall performance of the video client. Besides that, we have discovered that, in a cognitive adaptive approach, bandwidth constraints affect the user's experience more substantially, while impairments such as packet loss can be prevented.**

***Keywords*—Adaptive Streaming, Reinforcement Learning, Wireless Networks, Android Applications**

I. Introduction

The last quarter of the 20th century saw the birth of wireless technologies. Skepticism reigned over the world and a situation in which a wireless device would be preferred against the fixed option was deemed unrealistic and only in the heads of a few optimistic. Nowadays, we know that not only this feeling was wrong but that the wireless growth is faster and more powerful than the most positive predictions. Furthermore, its development is far from saturation. It is expected that by 2020 around 50 billion devices will be connected in a world wide network [1]. In parallel to this development, video content is nowadays a very significant portion of the Internet traffic. Cisco predicts that by the end of 2015 more than 65% of the mobile traffic will be accounted to Video Streaming Services [2]. Thus, adapting Video Streaming Services to the demands of mobile devices in an ever growing wireless network becomes fundamental.

Dynamic Adaptive Streaming over HTTP for MPEG (MPEG-DASH) is one of the best known standards for Video Streaming applications over the Internet. This technique's success relies on continuously adapting the bit-rate as the video is streamed over the network, so that the quality in the client's side is maximized [3]. Defining how to measure quality and choosing the segment selection algorithm are challenges that have been approached by different research paths. Quality assessment on the client side has traditionally been a task assigned to Quality-of-Service (QoS) parameters such as packet losses, delays or bandwidth. However, when dealing with wireless networks, QoS-based control is mostly inaccurate and insufficient [4] due to the fact that its metrics reflect the status of the network but do not capture the quality delivered and perceived by the end-user, i.e. the user's Quality of Experience (QoE) [5]. A more effective bit-rate selection algorithm should be able to adapt to different and unpredictable non-deterministic conditions of the network [6], [7]. In this regard, Machine Learning and in particular Reinforcement Learning (RL) techniques have proven to fit well in adaptive streaming applications. Examples of this are the State-Action-Reward-State-Action (SARSA) approach of Menkovski et al. [8] or the Q-Learning algorithm of Claeys et al. [9], [10], which have been studied on desktop computers. Herein, we intend to assess the viability of RL-based adaptive streaming in the context of constrained mobile devices, looking particularly at Android-based terminals.

In this research, we present a novel smart video player client in Java for Android using MPEG-DASH. Based on the Q-Learning approach presented by Claeys et al. [9], we have designed and implemented an Adaptive Video Streaming Application fit for running in Android-based mobile devices. By means of a self-developed wireless experimental test-bed we evaluated the performance of the application under a wide range of conditions. In this way we could pinpoint the network impairments that affect QoE the most in close-to-real network situations. Through this analysis we have been able to improve the overall performance of the algorithm by including in the reward function contributions regarding downloading speed and already buffered video segments.

The remainder of this paper is organized as follows. Section II provides with background on concepts like Reinforcement Learning and specifically Q-Learning, state-of-art

on Adaptive Streaming technologies and on the Android design and architecture. Section III, gives a thorough explanation of our Adaptive Streaming learning algorithm. The wireless test-bed and android application development insights are presented in section IV. Experiments and results can be found in section V-B. Finally, section VI draws some conclusions highlighting contributions and presents directions of future work.

II. Background

In this section, we present background knowledge useful for the non-specialist to understand the remaining of this paper. Firstly, we introduce the Reinforcement Learning framework (section II-A). HTTP Adaptive Video Streaming techniques in general and MPEG-DASH in particular are presented in section II-B. Finally, section II-C discusses the Android OS, its architecture and advantages for video streaming.

A. Reinforcement Learning and Q-learning

Reinforcement Learning (RL) [11] is the field of Machine Learning (ML) inspired by psychology, which studies how artificial agents can perform actions to achieve a specific goal. The agent controls a dynamic system by choosing actions in a sequential order. The dynamic system, also known as environment, is characterized by its states, its dynamics, and a function describing the state's evolution given a group of actions chosen by the agent. After executing one action, the agent moves to a new state, where it receives a reward (scalar value) informing how far it is from its goal (the final state). To achieve the goal, the agent has to learn a strategy to select actions, policy in literature, in such a way that the expected sum of the rewards is maximized over time.

In this paper we focus on the Q-learning [12] algorithm, which is a model-free RL technique suitable to model the environment in the context of HTTP Adaptive Video Streaming. It has a function which calculates the quality of a state-action combination, $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Initially, before learning, the Q function returns any arbitrary fixed values, selected by the designer, denoted by policy π. Afterwards, each time t that the agent selects an action a_t in a given state s_t, it observes a reward R_{t+1} and a new state s_{t+1}. These observations are used to update the Q function. In the end, the agent will learn an optimal policy π^* which will give to it, the possibility to chose the best action in a given state to fulfill its goal. The learning update rule for the Q-Learning algorithm is given by:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s, a)\big[\mathcal{R}_{t+1} + \gamma \max Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\big] \quad (1)$$

where R_{t+1} is the reward observed after performing action a_t in state s_t, and where $\alpha_t(s, a)$ is the learning rate, with all $\alpha \in [0, 1]$. Learning the action a chosen for a specific state s can be done using the $\epsilon - greedy$, $\epsilon - soft$, or softmax strategies [13]. After learning, the strategy to select the best action a is given by the optimal policy π^* as it is shown next:

$$\pi^*(s) = arg\max_a Q^*(s, a), \forall s \in S \quad (2)$$

B. Dynamic Adaptive Streaming over HTTP

Video Streaming Services have been gradually growing in popularity to account, nowadays, for 65% percent of the global mobile data traffic [2]. They can be classified into two categories, Internet Protocol Television (IPTV), offered by network providers, and Over-The-Top (OTT) services, group in which HTTP Video Streaming Applications are included. A video server hosts videos split in small segments (usually between 2 and 10 seconds) and encoded to different quality levels. During a streaming session and based on the network conditions (e.g. packet loss, jitter, bandwidth availability), the client chooses to download a video segment at a specific quality, while trying to maximize the user's experience. An additional benefit given by this technique, is that, due to the HTTP protocol characteristics, packet losses will not affect the quality of the user's experience, except in severe network outages cases.

In the early stages of its development, industry took charge and several HTTP Adaptive Streaming solutions, such as: IIS Smooth Streaming by Microsoft [14], HTTP Live Streaming by Apple [15], and Adobes HTTP Dynamic Streaming [16] were developed. MPEG-DASH was born as an alternative to the industrial proprietary streaming techniques. Figure 1 shows the DASH architecture [17]. DASH structures multimedia content in a hierarchical fashion. The top level describes an entire media object (the video) and is called a Presentation. A Media Presentation Description (MPD) describes the several objects comprised in a Presentation. The bottom of the hierarchy consists of video segments. There are additional structures used to select among and within segments to achieve various playback requirements. The selection may be based on a number of factors and techniques. One of the possible selection techniques is the RL-approach.

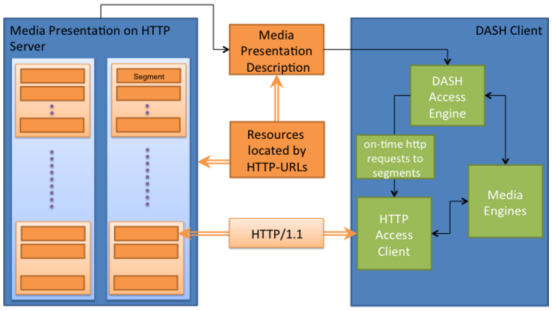

Fig. 1: MPEG-DASH High-level architecture

The idea behind the use of RL for MPEG-DASH comes from the ease to map the streaming technology to a model-free RL problem. The states space $\mathcal{S}$ is given by the Cartesian product between varying network conditions measured on the client side and client characteristics (e.g. download speed, buffer filling). Despite the continuous nature of the states space, discretization methods can be applied to it to make it suitable for discrete RL algorithms. The actions space $\mathcal{A}$ is given by the different level of quality (i.e. bit-rate) encoded for each video segment, and the reward function $\mathcal{R}_{\cdot}(.,.)$ by a linear combination between different components (buffer filling, bandwidth, segment's quality). Thus, using the RL approach, the QoS factors and the precoded subjective perception of

videos from people can be combined to maximize the user's experience in Video Streaming Application.

C. Android OS

Android is an OS for mobile devices and currently developed by Google. It is architected in the form of a software stack comprising applications, an operating system, run-time environment, middle-ware, services and libraries, as represented in figure 2 [18].

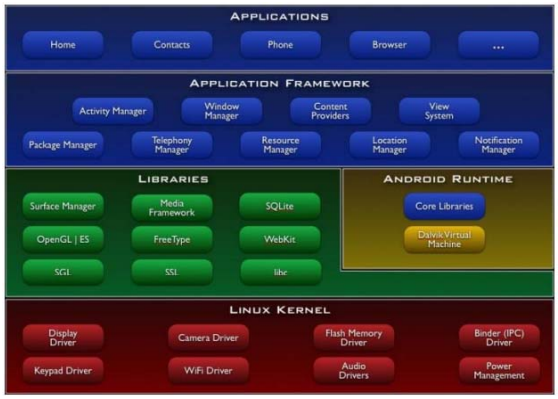

Fig. 2: Android Stack Architecture

Android fundaments itself on a Linux kernel, layer one, which introduces a level of abstraction between the device hardware and the upper layers.

Each application on an Android device runs as a process within its own instance of the Dalvik virtual machine (VM) (layer two), thus avoiding interference among applications or with the OS.The Android Core Libraries, Also located in layer two, fall into three main categories. Firstly, the Dalvik VM Specific libraries interact with the VM instances. Secondly, the Java Interoperability library, open source and developed predominantly in Java, are employed for tasks such as string handling, networking and file manipulation. Lastly, the Android libraries, also java-based but specific to Android, include from framework to libraries to facilitate user interface building, graphics and media. Apart from the Android Runtime VM and libraries, the layer two comprises C/C++ libraries which interact with the kernel to fulfill a wide and diverse range of functions including 2D and 3D graphics, Secure Sockets layer (SSL) communication or audio and video playback. Thus, these libraries are fundamental for the development of our adaptive streaming application. They can be accessed through the Java based Android core libraries.

The Application Framework is a set of services that collectively form the environment in which Android applications run and are managed. These applications, located on top of the stack, comprise both native and third party applications installed by the user.

III. Adaptive Streaming Q-Learning Algorithm

In this section, we present our algorithm developed for the Adaptive Streaming Application for Android, based on the Q-Learning algorithm discussed in section II and adapted to the Android environment.

Algorithm 1 Adaptive streaming Application Algorithm

while $t \in$ Streaming Session **do**
  Calculate s_t from system's parameters at time $t-1$
  Obtain $a_t = \max_{z \in |\mathcal{Q}|} Q_t(s_t, z)$
  Download video segment v_t with quality a_t
  Calculate s_{t+1} from system's parameters at time t
  Calculate $R_{t+1}(s_t, a_t)$ according to equation 3
  Update $Q_{t+1}(s_t, a_t)$ according to equation 1
end while

Algorithm 1 shows the basic functionality of the streaming algorithm. During a session, when a new segment (v_t) is to be downloaded, the current state (s_t) is computed from the previous system's parameters. After that, by picking the best possible action a_t ,i.e. the most suitable quality of a video segment at time t according to the current knowledge, i.e. the Q values obtained from the previous iteration (Q-table), the new segment is downloaded. Furthermore, the reward function ($R_{t+1}(s_t, a_t)$) is computed and the Q-table is updated. The reward function, $R_{t+1}(s_t, a_t)$ (equation 3), is composed by contributions ranging from quality correlation to downloading speed. Each of the contributions is multiply by a constant (C_1 - C_6) according to its weighted importance on the overall reward. $R_{t,q}(s_t, a_t)$, $R_{t,o}(s_t, a_t)$, $R_{t,bf}(s_t, a_t)$ and $R_{t,bc}(s_t, a_t)$ are based on the reward function derived by [9]. $R_{t,bs}(s_t, a_t)$ and $R_{t,ds}(s_t, a_t)$ are our own contribution and will be thoroughly discussed. These two reward contributions improve drastically the performance of the algorithm, as it will be demonstrated in section V-B.

$$\begin{aligned} R_{t+1}(s_t, a_t) = {} & C_1 * R_{t,q}(s_t, a_t) + C_2 * R_{t,o}(s_t, a_t) \\ & + C_3 * R_{t,bf}(s_t, a_t) + C_4 * R_{t,bc}(s_t, a_t) \\ & + C_5 * R_{t,bs}(s_t, a_t) + C_6 * R_{t,ds}(s_t, a_t) \end{aligned} \quad (3)$$

The quality reward, $R_{t,q}(s_t, a_t)$, focuses on the quasilinear correlation between the QoE objective PSNR values and the subjective MOS evaluations [19]. Equation 4 shows the relation between the streaming quality level chosen for interval t (a_t) and the number of qualities in the quality set $\mathcal{Q}$ ($|\mathcal{Q}|$).

$$R_{t,q}(s_t, a_t) = \frac{a_t - 1}{|\mathcal{Q}| - 1} * 2 - 1 \quad (4)$$

The oscillation reward, $R_{t,o}(s_t, a_t)$ (Equation 5), is based on the knowledge that a fluctuation in the quality level influences negatively the QoE. To model the oscillation, the authors of [9] defined the length and the depth of the oscillation, where length (OL_t) is the number of video segments since the last observation and depth (OD_t) is the quality difference before and after the oscillation. OL_{max} is the maximum length observed, which means that an oscillation of length higher or equal than OL_{max} will receive an oscillation reward of 0.

$$R_{t,o}(s_t,a_t) = \begin{cases} 0 & \text{:no oscillation} \\ \frac{-1}{OL_t^{\frac{2}{OD_t}}} + \frac{OL_{i-1}}{(OL_{max}-1) * OL_{max}^{\frac{2}{OD_t}}} & \text{:oscillation} \end{cases} \quad (5)$$

$R_{t,bf}(s_t, a_t)$ and $R_{t,bc}(s_t, a_t)$ (equations 6 and 7) refer to the importance of buffer starvations when rewarding or

penalizing the Q learned value. Buffer starvations lead to video freezes and these have a considerable effect on QoE. Equation 6 focuses on the buffer filling level, penalizing with a maximum -1 when the buffer level B_t in interval t is below 10% of the buffer size B_{max}. A linear function for the interval of $[-1, 1]$ provides the rewards of a buffer filling below that threshold. In addition, equation 7 analyses the influence of the changes in the buffer level between intervals, rewarding or penalizing the buffer filling change when the filling is low.

$$R_{t,bf}(s_t,a_t)=\begin{cases} -1 & :B_t \leq 0.10 * B_{max} \\ \frac{2*B_t}{(1-0.1)*B_{max}} - \frac{1+0.1}{1-0.1} & :B_t > 0.10 * B_{max} \end{cases} \quad (6)$$

$$R_{t,bc}(s_t,a_t)=\begin{cases} \frac{B_t - B_{t-1}}{B_{t-1}} & :B_t \leq B_{t-1} \\ \frac{B_t - Bt-1}{B_t - \frac{Bt-1}{2}} & :B_t > *B_{t-1} \end{cases} \quad (7)$$

Looking for ways to enhance the learning speed of the algorithm and its accuracy to a varying wireless channel, we have developed and included two new elements on the reward function. Operating in a lightweight environment, as the Android OS, the biggest influence in the quality comes in the form of video freezes. These freezes appear not only due to buffer starvations but also because of the changing conditions on the transmission channel. Thus, for the development of these two reward elements we focused on how the fluctuations in the channel would affect the segment in two aspects: its downloading speed and its presence in the buffer before it is to be displayed.

The buffered segment reward, $R_{t,bs}(s_t, a_t)$ (equation 8), rewards or penalizes the system depending on the segment being previously buffered. N_b, N_t, N_b^{max} and a_t are number of buffered segments, current segment, maximum number of segments to be buffered and current quality respectively.

$$R_{t,bs}(s_t, a_t) = \frac{((4 - (N_b - N_t) * (N_b^{max} - a_t)}{N_b^{max}} \quad (8)$$

Algorithm 2 $R_{t,ds}(s_t, a_t)$ calculation

Step 1: Calculate average downspeed (DS^{av})
$DS^{av} = (1/N_{seg}) * (\sum_{i=t-N_{seg}}^{t} DS_i)$
Step 2: Estimate download time ($E(s_t, a_j)$)
$E(s_t, a_t) = \frac{SIZE(s_t,a_t)}{DS^{av}}$
Step 3: Calculate $R_{t,downspeed}$

$$R_{t,downspeed}(s_t, a_t) = \begin{cases} \frac{-E(s_t,a_t)}{D_1} & :E(s_t, a_t) > DUR(s_t) \\ \frac{+E(s_t,a_t)}{D_1} & :E(s_t, a_t) \leq DUR(s_t) \end{cases}$$

The downloading speed reward, $R_{t,ds}(s_t, a_t)$ (algorithm 2), takes charge of the task of either rewarding the algorithm if the segment downloading speed is accurate to the chosen quality or penalizing if on the contrary it is too low. On the first stage of the algorithm the download speed of the last period is averaged. N_{seg} is the number of segments corresponding to the time window set to analysis. Its value depends on the conditions of the network and has empirically been set to 15. In the second step, the download time for the next segment is estimated $E(s_t, a_t)$. Finally, the algorithm is rewarded if the estimated time is shorter than the segment duration and penalized otherwise.

IV. Experimental Test-bed

Herein, we give an overview of the test-bed used for the experimental analysis. Furthermore, we provide some insight on the Android application developed.

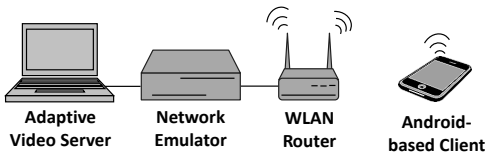

Fig. 3: Experimental setup

Figure 3 shows our experimental set-up. The video test-set is saved in a computer running a HTTP Apache server. The computer is connected to one of the interfaces of the network emulator (PacketStorm Hurricane II) whose purpose is to emulate network conditions like packet losses and bandwidth throttles. The emulator is connected at the same time to a standard IEEE 802.11 wireless router through one of its ethernet interfaces. The Adaptive Streaming Application running in the client, an Android-based Samsung tablet, connects to the Apache Video server through the wireless link of the router.

When the user starts the Q-Learning based Adaptive Streaming Application, the client tries to connect to the server using the HTTP protocol. Once communication is established, the client requests the list of available videos. On reception, the list is displayed in the application interface and the user can select one of the videos availables in the server. Clicking on the video triggers the starting of the streaming session in the client's display. This is done by an information request from the client to the server. The video information is used to start the selection algorithm. This algorithm chooses the most appropriate segment to be downloaded as explained in section III and requests the video to the server accordingly. New segments will be continuously requested from the server and downloaded to the client as long as the streaming session takes place. On session termination, statistics graphs, such as Q value's evolution, rewards or quality oscillations are displayed on user's demand. At the beginning of the session the application will always request the lowest quality. This is a defense mechanism to avoid freezes. As soon as the client starts calculating its Q values and rewards, the quality will be better adapted to the available capacity and network conditions.

Regarding the software tools used, the application has been developed in Java for Android. We have used the inbuilt libraries for implementing the HTTP communication between client and server (URL and HTTP), for the segment display (media), graphics, etc.

V. Experiments and Results

In this section we first provide a short description on the video set employed in the experiments and discuss the parameters and constants values used for the Q-Learning

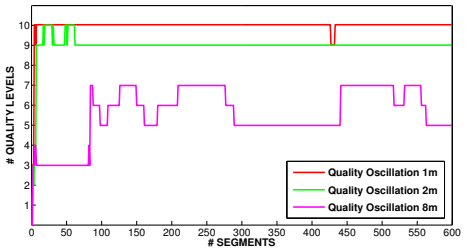

Fig. 4: Results Distance to the Wireless Access Point.

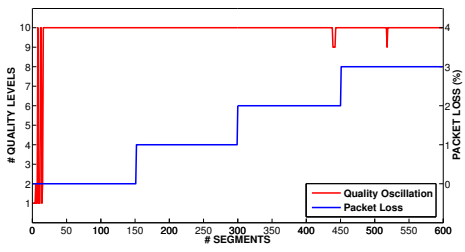

Fig. 5: Results Network Impairments: Packet Losses

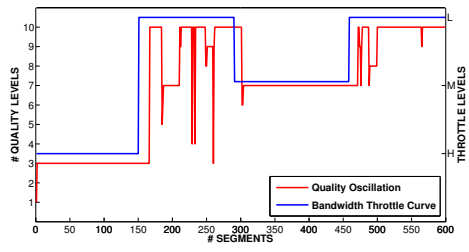

Fig. 6: Results Network Impairments: Bandwidth Throttle

algorithm (section V-A). These conditions and parameters are then used in the experiments showed in section V-B.

A. *Video test-set and Environmental state*

The video test-set used for the experiments consist of 5 segments 2 seconds long of the video Shields from the Live Video Database [20]. The video has been segmented and encoded in mp4 to 10 different qualities between 64 kbps to 2048 kbps as it can be seen in table I.

Table II shows the constant values used for the experiments. C_1 to C_4, α and γ are based on the values given in [9]. Through experimental analysis we set C_5 and C_6 to 1. N_{seg} is 15 understanding that in a changing environment such as the wireless channel, monitoring for a period of 30 seconds prior to the download gives us enough information about the evolution of the network. D_1, the divisor for the downloading speed reward, was experimentally set to 6.

Quality	bitrate
1	64 kbps
2	128 kbps
3	256 kbps
4	384 kbps
5	512 kbps
6	640 kbps
7	768 kbps
8	1024 kbps
9	1556 kbps
10	2048 kbps

TABLE I: Quality level bitrates for the Shields video traces

Constant	Value
α	0.3
γ	0.95
C_1	1
C_2	1
C_3	2
C_4	2
C_5	1
C_6	1
D_1	6
N_{seg}	15

TABLE II: Constant values used for the experiments

B. *Experiments*

The purpose of this battery of tests is firstly to evaluate how our adaptive video streaming application performs in the presence of the different conditions derived from dealing with a wireless channel. We define performance as the time, or number of segments, that the algorithm takes until converging to the optimal quality given the surrounding conditions. Through this analysis we aim to pinpoint impairments or conditions to which our learning algorithm is more vulnerable to.

1) Distance to the Wireless Access Point: The aim of this first experiment is to evaluate the Android application in optimal conditions. Thus, in this case the network emulator won't be introducing further impairments. As explained in the beginning of this section, our second aim is to understand how different physical conditions would affect the application, and so, making it take longer to converge and eventually reducing the user's QoE. For this reason, in this first set of test, we decided to analyse the performance while studying the effect of the distance to the access point on the received quality.

Figure 4 shows the results of the 20 minutes streaming sessions performed at three different distances: 1 m, 2 m and 8 m. From these results it can be seen that the closer to the wireless access point, the faster the algorithm converges to a quality. While at 1 m distance the algorithm takes only 10 to 20 segments to converge, for the 2 m distance nearly 70 segments are needed. This effect is even more evident in the 8 m case, where the 600 segments of session are not enough for the application to converge for an optimal quality. Another effect that can be seen is that the application converges to a lower quality level as the distance increases. Maximum quality level (10) for the closest point, 9 when the distance doubles and $6-7$ with a distance to the client of 8 m.

From this first round of tests we concluded, that our algorithm is able to converge to the optimal quality level in a short number of segments. The contribution of the downloading speed and already buffered segments rewards become fundamental in this fast convergence. Furthermore, we could also assess the effect on the access point-client distance on both the convergence time and the chosen quality.

For the next two experiments, we set the client in the closest position, and its performance as our benchmark to compare with, when further network impairments are included.

2) Network Impairments-Packet Losses: Packet losses have been demonstrated to affect greatly in video streaming applications [21]. Thus, this second experiment.

We set the Android-client at the distance of 1 m from the router and we start the 20 minutes streaming session. With the help of network emulator we gradually increase the network packet losses from 0% to 3% in a 5 minute interval.As it can be seen in figure 5, at the beginning of the streaming session with no packet losses, the algorithm takes $20-25$ segments to converge to the maximum quality. Counter intuitively, as the packet losses increase (blue line) the effect in the quality oscillation is non existent and the quality is maintained to the maximum during the rest of the session. The reason for this is rooted on the HTTP protocol retransmission policy. Packets that are lost are retransmitted fast enough for the algorithm not to notice them.

From this, we understood that network packet losses won't

affect the HTTP Video streaming application, except in extreme network autages.

3) Network Impairments-Bandwidth Throttle: In our last tests we focused on how our application would react to restrictions on the bandwidth. Thus, again in the network emulator we set three different limits in the bandwidth: Low (L, 4096 kbps), Medium (M, 1024 kbps) and Hard (H, 512 kbps).

Figure 6 shows the results of this last experiment. The 20 minutes streaming session is divided in 5 minutes sections, each of them limited by one of the previously defined bandwidths. The first 5 minutes (H limit), the application learns to adapt to the quality value below the threshold in 5 to 10 segments. At time 5min the limitation is reduced to its lowest value (L) and the application tries to adapt to the new status. However, due to the information already learned, the application's quality oscillates for some segments until finally adapting to the maximum quality level. Same performance can be observed at times 10min and 15min when the limits are changed from L to M and from M to L.

These results make us conclude that limiting the bandwidth reserved for the streaming session affects the performance of the application in a substantial manner and should be taken into account when improving the reward function.

VI. Conclusion

In this work we have presented a novel MPEG-DASH Adaptive Video Streaming Application in Java for Android. Inspired by a RL-Q learning approach we have designed and implemented an application fit for running in Android devices. By means of a self-developed wireless experimental test-bed we have evaluated the application and improved its performance by including information on the segments' downloading speed and their presence in the buffered for the rewarding scheme. These two contributions become fundamental to achieve fast convergence time and improved performance needed for the ever changing wireless channel. Furthermore, we have pinpointed network conditions which would affect the most in a real network situation. We have seen that while packet losses are avoided thanks to the HTTP protocol scheme, bandwidth throttles and distances to the access point are the predominant factors that affect QoE.

These results give us further ideas about possible improvements on the reward function, such as including knowledge and prediction about the network or device characteristics (battery level, CPU load, screen resolution).

Acknowledgment

Funding from the ARTEMIS project DEMANES (Design, Monitoring and Operation of Adaptive Networked Embedded System - Grant 295372) and the European Research Council project BROWSE (Beam-steered Reconfigurable Optical-Wireless System for Energy-efficient communication - Grant 291632) are gratefully acknowledged.

References

[1] A. Liotta, "The cognitive NET is coming," *IEEE Spectrum*, vol. 50, no. 8, pp. 26–31, Aug. 2013.

[2] Cisco, "VNI forecast," http://ciscovni.com/vniforecast/index.htm.

[3] T. Stockhammer, "Dynamic adaptive streaming over http –: Standards and design principles," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943572

[4] M. Torres Vega, S. Zou, D. C. Mocanu, E. Tangdiongga, A. M. J. Koonen, and A. Liotta, "End-to-end performance evaluation in high-speed wireless networks," in *The International Conference on Network and Service Management (CNSM)*, 2014.

[5] J. Klaue, B. Rathke, and A. Wolisz, "Evalvid - a framework for video transmission and quality evaluation," in *In Proc. of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 2003, pp. 255–272.

[6] G. Exarchakos, L. Druda, V. Menkovski, P. Bellavista, and A. Liotta, "Skype resilience to high motion videos." *IJWMIP*, vol. 11, no. 3, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijwmip/ijwmip11.html#ExarchakosDMBL13

[7] A. Liotta, "Farewell to deterministic networks," in *IEEE 19th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, Nov. 2012.

[8] V. Menkovski and A. Liotta, "Intelligent control for adaptive video streaming," in *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, Jan 2013, pp. 127–128.

[9] M. Claeys, S. Latré, J. Famaey, and F. De Turck, "Design and evaluation of a self-learning http adaptive video streaming client," *Communications Letters, IEEE*, vol. 18, no. 4, pp. 716–719, April 2014.

[10] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design of a q-learning-based client quality selection algorithm for http adaptive video streaming," in *Adaptive and Learning Agents Workshop, part of AAMAS2013, Proceedings*, 2013, pp. 30–37.

[11] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. Springer, 2012, vol. 12.

[12] C. J. C. H. Watkins and P. Dayan, "Technical note: q-learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[14] "Microsoft. Smooth Streaming: The Official Microsoft IIS Site," http://www.iis.net/downloads/microsoft/smooth-streaming.

[15] R. Pantos and W. May, "HTTP Live Streaming," http://tools.ietf.org/html/draft-pantos-http-live-streaming-10.

[16] "Adobe. HTTP Dynamic Streaming: Flexible delivery of on-demand and Live Video Streaming," http://www.adobe.com/products/hds-dynamic-streaming.html.

[17] Qualcomm, "Dynamic Adaptive Streaming over HTTP (DASH)," last Accessed November 2014.

[18] "An Overview of the Android Architecture," http://www.techotopia.com/index.php/An_Overview_of_the_Android., last Accessed: 9th December 2014.

[19] O. Nemethova, M. Ries, M. Zavodsky, and M. Rupp, "PSNR-Based Estimation of Subjective Time-Variant Video Quality for Mobiles," in *In Proc. of MESAQIN 2006*, 2006.

[20] K. Seshadrinathan, R. Soundararajan, A. C. Bovik, and L. K. Cormack, "Study of subjective and objective quality assessment of video," *Trans. Img. Proc.*, vol. 19, no. 6, pp. 1427–1441, Jun. 2010.

[21] D. Mocanu, A. Liotta, A. Ricci, M. Vega, and G. Exarchakos, "When does lower bitrate give higher quality in modern video services?" in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–5.