D-DASH: A Deep Q-Learning Framework for DASH Video Streaming

Matteo Gadaleta, Federico Chiariotti, *Student Member, IEEE*, Michele Rossi, *Senior Member, IEEE*, and Andrea Zanella, *Senior Member, IEEE*

***Abstract*—The ever-increasing demand for seamless high-definition video streaming, along with the widespread adoption of the dynamic adaptive streaming over HTTP (DASH) standard, has been a major driver of the large amount of research on bitrate adaptation algorithms. The complexity and variability of the video content and of the mobile wireless channel make this an ideal application for learning approaches. Here, we present D-DASH, a framework that combines deep learning and reinforcement learning techniques to optimize the quality of experience (QoE) of DASH. Different learning architectures are proposed and assessed, combining feed-forward and recurrent deep neural networks with advanced strategies. D-DASH designs are thoroughly evaluated against prominent algorithms from the state-of-the-art, both heuristic and learning-based, evaluating performance indicators such as image quality across video segments and freezing/rebuffering events. Our numerical results are obtained on real and simulated channel traces and show the superiority of D-DASH in nearly all the considered quality metrics. Besides yielding a considerably higher QoE, the D-DASH framework exhibits faster convergence to the rate-selection strategy than the other learning algorithms considered in the study. This makes it possible to shorten the training phase, making D-DASH a good candidate for client-side runtime learning.**

***Index Terms*—DASH, HTTP adaptive streaming, reinforcement learning, deep Q-learning, LSTM.**

I. Introduction

VIDEO streaming has been the dominant source of Internet traffic for the last few years; right now, videos make up 55% of all mobile traffic, and this figure is predicted to rise to 75% in the next five years [1].

Since its inception in 2011, Dynamic Adaptive Streaming over HTTP (DASH) [2] has become the dominant standard for video transmission, as it relies on the existing HTTP server and Content Delivery Network (CDN) infrastructure and is not affected by firewalls and NATs. The DASH standard leaves complete freedom to the client in the choice of the adaptation policy: videos are divided into short segments (usually a few seconds long), which are encoded at different compression levels to generate an *adaptation set* of *representations* at different bitrates. The server makes all the segments in the adaptation set available to the client as HTTP resources, as well as a Media Presentation Description (MPD) file containing all the information about the video segments and their URLs. The client sequentially downloads each segment, switching between representations according to its adaptation logic in an attempt to optimize the Quality of Experience (QoE) for the current video and network conditions.

The research on DASH adaptation algorithms is still ongoing, and most commercial systems employ very basic heuristic approaches, leading to annoying quality variations [3], [4] and to an inefficient use of network resources. In order to maximize the user QoE, the adaptation logic needs to take into account both the video content, which affects the perceived quality of the downloaded representations, and the playout buffer state. In fact, a major factor in video QoE is rebuffering [5], i.e., the temporary freezes in the video playout as the client waits until the next segment is downloaded [6].

In this challenging scenario, Reinforcement Learning (RL) has emerged as an elegant and viable solution to the video adaptation problem. RL-based algorithms learn from past experience by trial-and-error, and gradually converge to the optimal policy [7]. The biggest design issue in these systems is that the state space of the corresponding Markov Decision Process (MDP) is very large. On the other hand, the number of states needs to be small to ensure quick convergence and make online solutions react timely to changes in the environment statistics.

In this work, we advocate the use of deep neural networks to learn excellent video adaptation strategies, while compactly and effectively capturing the experience acquired from the environment. The use of deep neural networks leads to several advantages, such as: i) the ability to deal with very large state spaces efficiently, effectively coping with the curse of dimensionality issue of RL algorithms, ii) the possibility of compactly representing the acquired experience through a set of weights, iii) the attainment of much better QoE performance, that is here quantified in terms of both the instantaneous visual quality of each segment and the quality variations across video segments, as well as the frequency of freezing/rebuffering events. We hence propose D-DASH, a framework for DASH video streaming that, combining deep neural networks with a carefully designed reinforcement

Manuscript received March 9, 2017; revised June 28, 2017 and September 11, 2017; accepted September 12, 2017. Date of publication September 21, 2017; date of current version December 22, 2017. This work has been supported by the University of Padova through the project CPDA 151221 "IoT-SURF: a unifying abstraction and reasoning framework for connected and unconnected objects". The associate editor coordinating the review of this paper and approving it for publication was D. Niyato. *(Corresponding author: Matteo Gadaleta.)*

The authors are with the Department of Information Engineering, University of Padova, 35131 Padova, Italy (e-mail: gadaleta@dei.unipd.it; chiariot@dei.unipd.it; rossi@dei.unipd.it; zanella@dei.unipd.it).

Digital Object Identifier 10.1109/TCCN.2017.2755007

learning mechanism, yields better QoE than prominent state-of-the-art rate-adaptation algorithms. More specifically, the main contributions of this paper are the following.

- We review existing RL-based techniques for video adaptation, emphasizing their limitations in terms of training effort (time and required memory space) and QoE.
- We formulate the DASH video streaming problem within a Deep Q-learning framework, detailing our design choices, that include: the choice of a reward function that effectively takes into account video quality variations and freezing/rebuffering events, the use of a learning architecture featuring two twin neural networks and a replay memory to get an improved stability and a faster convergence, and the use of a pretraining phase (on synthetic traces) to speed up the convergence of D-DASH when confronted with real traces.
- We evaluate deep forward and recurrent neural network architectures and assess their training cost (time and memory) and their attained QoE.
- We provide a thorough numerical validation of D-DASH, considering real and simulated traces and comparing it against prominent algorithms from the state-of-the-art.

Our results shed some light on the importance of tracking a certain amount of channel memory in the learning architecture, especially for complex network scenarios, and the superiority of the proposed deep Q-learning designs: in all our experiments, these have shown a higher average quality with smaller fluctuations across video segments, and a much lower percentage of rebuffering events.

The rest of the paper is organized as follows: Section II presents the state of the art in DASH adaptation logic, with a focus on RL-based algorithms. Section III defines the system model, detailing the decision making instants, the playout buffer behavior and the reward function. Section IV uses these notions within a decision making framework, starting from a Markov decision process approach to then delve into standard and deep Q-learning based ones. Section V provides some preliminaries on the neural network structures that will be considered in our framework and Section VI presents the D-DASH deep Q-learning system. Section VII describes the simulation settings/parameters, quantifies the performance of D-DASH over synthetic and real channel traces, and compares it against selected algorithms from the literature. Finally, Section VIII concludes the work.

II. Related Work

In this section, we provide an overview of the most relevant works on DASH client-side adaptation strategies in the literature. As measuring QoE itself is a subject of intensive research in the field, there is not a single set of reference performance metrics [8]. In-depth reviews of the factors that impact QoE and the way they are measured in different streaming systems can be found in [9] and [10]. Among such elements, in this work we focus on the following three factors. The first, and most obvious, is the *instantaneous picture quality*, which can be assessed through different techniques. Notable ones are: objective metrics such as the bitrate, no-reference metrics, which gauge the video distortion solely from the received frames (i.e., no external quality reference is provided) [11], or full-reference metrics such as Structural Similarity Index (SSIM) [12], a perception-based metric which measures the distortion between the input and output of the video encoder at the transmitter side. These approaches are adopted by various DASH adaptation algorithms in [10], [13], and [14]. The second factor, which is often the most pressing concern in adaptive video streaming, is represented by *rebuffering events*: their length and frequency strongly affect user QoE, as demonstrated by Hoßfeld *et al.* [5]. The third factor is *video quality stability*: users notice frequent transitions in the video quality and might be annoyed by them [15]; in DASH video streaming, quality changes are not due to packet loss, but to switches between different adaptations (i.e., sudden changes in the quality of the video).

The literature on adaptation logics for DASH is vast, and we refer the reader to [16] for a comprehensive overview of existing techniques. For the purpose of this work, adaptation logics can be divided into two broad categories: 1) heuristics and 2) dynamic programming based. One of the earliest adaptation algorithms to go beyond simple buffer-based control loops was the Fair, Efficient, and Stable adapTIVE algorithm (FESTIVE) [17]: this scheme does not address QoE explicitly, but rather uses a stability cost function and a limit on the frequency of bitrate increases to privilege stability over instantaneous video quality. FESTIVE has the added advantage of being fairer than commercial protocols in terms of QoE, as well as of reducing the number of rebuffering events because of its conservative approach when increasing the bitrate.

Probe and Adapt (PANDA) [18] is another landmark algorithm in the DASH adaptation literature. It uses a proactive probing strategy to evaluate the channel capacity and changes its rate estimate according to an additive-increase approach to prevent fluctuations due to non-persistent cross-traffic and on-off effects. In stationary conditions, its bitrate estimate equals the TCP throughput. PANDA then uses a hysteresis threshold to avoid frequent switches between adjacent representations. A similar but simpler heuristic, called QoE-driven Rate Adaptation Heuristic for Adaptive video Streaming (QoERAHAS), was presented by Petrangeli *et al.* in 2014 [19].

Both PANDA and FESTIVE tend to be extremely conservative in their decisions, and they often end up underutilizing the available capacity unless the network conditions are extremely stable. Heuristics are hard-pressed to efficiently exploit the available capacity and still limit rebuffering events and quality switches, unless they are designed with knowledge of the network statistics, thus enabling the adoption of a dynamic programming approach to optimally solve the adaptation problem.

Li *et al.* use finite-horizon dynamic programming [18], modeling the adaptation problem as a fairness problem between future segments, with an added penalty for quality switches. Adding the adaptation logic on top of PANDA's switching strategy, the authors manage to significantly improve video quality without losing the algorithm's buffer and quality stability properties. In order to reduce the computational load, the authors assume that the available capacity remains constant throughout the optimization horizon. However, this assumption could prove to be unrealistic in wireless channels, which can suffer from outages and quick capacity variations due to mobility, fading and cross-traffic.

Yin *et al.* [20] developed a system that uses Model Predictive Control (MPC) to make decisions based on a look-ahead approach, with a QoE model similar to that considered in this work. More specifically, they rely on a throughput predictor to make optimal choices for a finite time horizon of five future segments. This approach is equivalent to dynamic programming and has the same critical dependency on the quality of the throughput predictor: if the predicted channel statistics are inaccurate, the adaptation logic will make suboptimal decisions.

An interesting paper by Bokani *et al.* [21] uses a Markov Decision Process (MDP) model to determine the optimal adaptation policy with dynamic programming. The main issue of this solution is the computational load: the model is too complex to be solved at runtime. The authors propose several solutions to mitigate the complexity issue, but the performance of these heuristics is either unsatisfactory or requires to store a large amount of offline computational results in the device's memory. A more recent work by Zhou *et al.* [22] proposes a pseudo-greedy heuristic to tackle the same problem, with the same kind of limitations. Other works use MDPs in more specific situations, such as streaming for multihomed hosts [23] and cloud-assisted adaptive streaming [24].

A. Reinforcement Learning and DASH

There are several works in the literature that use RL to overcome dynamic programming's two biggest drawbacks: computational load, and the need to know the statistics of the network and video content in advance. RL-based algorithms learn the network statistics from experience, and their computational complexity is low. The main limitation of RL is the amount of experience it needs to make good decisions: as the number of states of the MDP grows, so does the necessary training time. Specifically, adaptive RL algorithms need to have a very coarse state granularity to effectively react to changes in the environment. Hence, there is a fundamental trade-off between adaptability and descriptive power: to be adaptable, the algorithm will need to visit and update each state frequently, i.e., to have a small number of states. However, having fewer states means that the knowledge of the environment is inevitably poorer (and, probably, so will be the algorithm's choices).

Two works by Claeys *et al.* represent the two opposite extremes in this trade-off: the algorithm in [25] has a complex reward function and a 6-dimensional state definition, and its training requirements are daunting, while the algorithm presented in [26] is lean and converges very quickly, but has a very rough state definition and a suboptimal performance. Another lean Q-learning algorithm that uses a similar MDP model was presented by Martín *et al.* in 2015 [27].

In 2015, van der Hooft *et al.* [28] presented an interesting hybrid between RL and standard algorithms: their system is based on Microsoft Smooth Streaming (MSS), but adapts the parameters of the heuristic to reflect the network conditions, improving its performance. The algorithm is still not QoE-aware, and the simple buffer-based MSS heuristic is suboptimal, but hybrid solutions such as this might be an interesting approach to overcome some of RL's main drawbacks.

In another recent work [29], the authors try to overcome the basic trade-off between efficiency and adaptability by parallelizing the learning process: since the problem has a well-known structure, experience in one situation can be used to learn how to act in others. This ad-hoc generalization scheme relies on the specific structure of the problem, but it is a step in the right direction.

As it will become apparent in the next sections, our D-DASH framework makes it possible to better exploit the observed data, learning from experience faster than other algorithms in the literature and quickly adapting the video encoding policy to the current working context.

III. SYSTEM MODEL

As we described in Section I, we consider a DASH client that downloads a video sequence segment by segment. The decision-making process follows a slotted time model, where $t = 1, 2, \dots$ is the video segment number. For any given segment t, the client is free to choose which representation to download from a given adaptation set A. Each representation, in turn, is uniquely associated with a certain video quality level of the segment, which we indicate by q_t. The aim is to find the policy that maximizes the QoE perceived by the user. In the following of this section we present our assumptions regarding the video streaming service and then we introduce the reward function that accounts for the QoE factors described in the previous section. Along the way, the main notation used in the paper will also be introduced.

A. Video Streaming Services

Each video segment is characterized by a certain quality-vs-rate trend, which we describe by means of the function $F_t(q_t)$ that gives the size of the segment t with quality q_t. We assume $F_t(\cdot)$ to be known before downloading segment t, as it can be made available to the client as part of the MDP or predicted from previous scenes, since the correlation between subsequent segments is usually high.

Denoting by C_t the average channel capacity experienced during the downloading of the segment t, we easily obtain the total downloading time τ_t as

$$\tau_t = \frac{F_t(q_t)}{C_t}. \tag{1}$$

We indicate by T the (constant) playout duration of a video segment. Moreover, we define the *buffer* (time) for segment t, denoted by B_t, as the lapse of time between the starting of the download of segment t and the instant the segment is due to start its playout at the client. Rebuffering events (during which the video playout freezes) occur whenever the playout buffer empties before the next segment has been completely downloaded, i.e., when $\tau_t > B_t$. Conversely, when $\tau_t < B_t$ the download of the segment t is completed before its planned playout time and the download of the next segment can start immediately, thus adding an extra $B_t - \tau_t$ time budget to the buffer of segment $t + 1$. Accordingly, the rebuffering time for a generic segment t is given by:

$$\phi_t = \max(0, \tau_t - B_t), \tag{2}$$

while the buffer for the next segment is computed as

$$B_{t+1} = T + \max(0, B_t - \tau_t). \tag{3}$$

The buffer is limited to 20 seconds, as most commercial video streaming systems limit video buffering to save memory and network capacity.

B. Reward Function

As discussed in Section II, the QoE of a video client depends on the visual quality of the current segment, the quality variation between segments, and the playout freezing events due to rebuffering. In the following, we introduce a reward function that captures these aspects and, in turn, can be used to derive policies that maximize the QoE of video streaming customers.

In this work, we chose SSIM as the instantaneous quality metric of a video sequence, so that q_t is the average SSIM of the video frames in segment t. SSIM is one of the most common objective metrics in the literature and has been shown to correlate well with the perceived QoE [30]. As it is a full-reference metric (i.e., its computation requires a full knowledge of the uncompressed segment [31]), it cannot be calculated by the streaming client, but its values when varying the video representation can be conveniently pre-computed, stored on the video server, and included as a field in the MPD. This puts the computational burden for calculating the SSIM onto the server side, even though, as reported in [32] the $F_t(q_t)$ characteristic of a video sequence can be automatically estimated from the size of the *encoded frames* by using a properly-trained deep neural network, thus making it possible to skip the computationally intensive frame-by-frame comparison with the original video sequence at the server. Other approaches are possible, for example computing the quality q_t of video frames at the client through a no-reference metric [11]. Although not considered in this paper, this is also viable and would require the implementation of an additional block at the client to estimate the quality of the received frames. The impact of inaccurate quality estimates, via no-reference metrics, is left to future investigations.

We finally define the reward function for segment t as follows:

$$\begin{aligned} &r(q_{t-1}, q_t, \phi_t, B_{t+1}) = \\ &= q_t - \beta\|q_t - q_{t-1}\| - \gamma\phi_t - \delta\big[\max(0, B_{\text{thr}} - B_{t+1})\big]^2. \end{aligned} \tag{4}$$

The first term on the right-hand side accounts for the benefit of a higher quality q_t of the video, while the following two negative terms are penalty factors due to *quality variations* in consecutive frames and *rebuffering events*, respectively. The right-most term is a further penalty that is applied whenever the buffer level is below a (preset) threshold B_{thr} and it has been introduced to further reduce the chance of highly-damaging rebuffering events. The coefficients β, γ, and δ are weighting factors that regulate the relative importance of the three penalty terms. Note that, neglecting the right-most penalty term (i.e., setting δ to zero), the reward function (4) is the same used in [20] and [29], and similar to that proposed and validated by De Vriendt *et al.* [15].

The weights β, γ and δ are here used to select different points in the trade-off between a high instantaneous quality, a constant quality level, and a smooth playback. The desired operational point might depend on several factors, including user preferences and video content, and tuning these parameters is outside the scope of this work.

IV. Machine Learning Optimization Framework

In this section, we present the machine-learning based approaches considered in this study to find video adaptation policies that try to maximize the perceived QoE. We start from an MDP formulation of the problem, and then describe the Q-learning and Deep-Q learning algorithms.

A. Markov Decision Process Model

If we model the video content as a sequence of scenes with exponentially distributed duration, like in [33], the adaptive video streaming service can be modeled as an MDP (see [34]), with action space A, state space S, and reward function $\rho: S \times S \times A \to \mathbb{R}$. By a slight abuse of notation, but for the sake of a compact notation, we use q_t to also indicate the action of downloading a segment t with visual quality q_t. The action $q_t \in A$, taken when the system is in a given state $s_t \in S$, determines the statistical distribution of the next state s_{t+1} and the reward $\rho(s_t, s_{t+1}, q_t)$ attained in step t. A policy is a function $\Pi : S \to A$ that maps states into actions. The *long-term utility* achieved by an admissible policy Π when starting from state s_0 is defined as

$$R(s_0; \Pi) = \lim_{h\to+\infty} \mathrm{E}\left[\sum_{t=0}^{h} \lambda^t \rho(s_t, s_{t+1}, \Pi(s_t)) \middle| s_0, \Pi\right] \tag{5}$$

where $\lambda \in [0, 1)$ is a discount factor that ensures convergence, and $P(s_{t+1}|s_t)$ is the one-step conditional transition probability of the state process $\{s_t\}$. An equivalent recursive formulation is given by:

$$R(s_0; \Pi) = \sum_{s_1 \in S} P(s_1|s_0)\rho(s_0, s_1, \Pi(s_0)) + \lambda R(s_1; \Pi). \tag{6}$$

The optimal policy $\Pi^*(\cdot)$ is finally found as:

$$\Pi^*(s) = \arg\max_{\Pi} R(s; \Pi), \quad \forall s \in S. \tag{7}$$

To apply this approach to our problem we need to map the action space, the state space, and the utility function to the parameters of our system. The action space clearly corresponds to the set A of possible representations of the video segments that can be chosen by the client. Accordingly, the action at step t corresponds to the choice of the quality q_t of the next segment to be downloaded. Therefore, from a state $s_t \in S$, an agent implementing a policy $\Pi(\cdot)$ will require the downloading of the segment t with quality $q_t = \Pi(s_t)$. The state space should be as small as possible, to reduce the complexity of the MDP, but at the same time each state should contain enough information to permit a precise evaluation of the utility function for each possible action $q_t \in A$. Considering the reward function (4) as a natural option for the utility function of the MDP, the state should hence include the video quality q_{t-1} of the last (the $(t-1)$-th) downloaded segment, the current buffer B_t state, the quality-rate characteristic $F_t(q_t)$ of the next segment (the t-th), and the future channel capacity C_t from which, given the chosen action q_t, it is possible to determine the download time τ_t of the next segment, the rebuffering time f_t and the next buffer state B_{t+1} using (1), (2), and (3), respectively. However, the future capacity C_t of the channel can only be known in a statistical sense, from the past observations. We then define the vector $\mathbf{C}_t = [C_{t-n}, C_{t-n+1}, \dots, C_{t-1}]$ of the previous n channel capacity samples, where n is assumed to be larger than the coherence time of the channel. In this case, the process $\mathbf{C}_t$ exhibits the Markov property, since the knowledge of samples that are further away in the past do not add any information to that contained in the current channel vector $\mathbf{C}_t$. Summing up, the state of the MPD at step t can be described by the 4-tuple $s_t = (q_{t-1}, \mathbf{C}_t, \phi_t, B_t)$.

Given the state s_t and the action q_t, we can finally define the utility function of our MDP as

$$\rho(s_t, s_{t+1}, q_t) = r(q_t, q_{t-1}, \phi_t, B_{t+1}). \tag{8}$$

The problem (7) can then be solved through dynamic programming, e.g., Value Iteration (VI) [34], but the computational complexity becomes rapidly unmanageable as the size of the problem grows. A possible approach to deal with the curse of dimensionality is to adopt RL tools, such as Q-learning [35], which gradually converges to the optimal solution through trial-and-error, as explained next.

B. Q-Learning

Q-learning is a RL algorithm introduced by Watkins and Dayan in 1992 [35]. It works by maintaining a table of estimates $Q(s, q)$ of the expected long-term reward (given by (6) in our problem formulation) for each state-action pair (s, q).

The Q-learning algorithm can use various exploration policies to decide the next action based on the Q-values: the most common are the ε-greedy policy and Softmax. Both strategies are non-deterministic; the ε-greedy policy chooses the action with the highest Q-value with probability $1-\varepsilon$, and a suboptimal action at random with probability ε. Softmax chooses an action according to the softmax distribution of Q-values:

$$p(q_t|s_t) = \frac{\exp\left(-\frac{Q(s_t,q_t)}{\xi}\right)}{\sum_{q\in A}\exp\left(-\frac{Q(s_t,q)}{\xi}\right)}, \tag{9}$$

where the parameter ξ sets the greediness of the algorithm. Greedier algorithms make suboptimal choices less often, but run a higher risk of getting stuck in local minima since they explore the state space less frequently.

In standard Q-learning, the Q-value $Q(s_t, q_t)$ is updated if the learning agent takes action q_t in state s_t. The future reward over the infinite time horizon is approximated as $\max_{q\in A} Q(s_{t+1}, q)$, i.e., the best Q-value of the future state s_{t+1}. If the Q-value matches the real expected long-term reward, Q-learning coincides with the Bellman formulation, which exactly solves the problem. In practice, since the Q-values are to be learned at runtime, they only provide an approximation of the real long-term rewards, but it has been proved that they converge to the optimal rewards for sensible exploration policies. The Q-learning update function is given by:

$$\hat{R}(s_t, q_t) = \rho(s_t, s_{t+1}, q_t) + \max_q \lambda Q(s_{t+1}, q) \tag{10}$$

$$Q(s_t, q_t) \leftarrow Q(s_t, q_t) + \alpha\left(\hat{R}(s_t, q_t) - Q(s_t, q_t)\right), \tag{11}$$

where the learning rate α sets the aggressiveness of the update and is usually decremented over time as the learning agent gets closer to convergence. The choice of the maximum Q-value in the bootstrap approximation (10) makes Q-learning an *off-policy* learning algorithm, since the greedy policy used in the long-term reward estimation (update policy) usually differs from the actual policy the learner uses (exploration policy). As the Q-values converge, the exploration policy should become greedier until it reaches convergence.

1) Limits of the Q-Learning Approach: The Q-learning approach is powerful, yet with some limitations: the algorithm provably converges to the optimal policy if its parameters are chosen correctly [35], but the convergence speed is an issue in complex problems. Kearns and Singh [36] proved that, for an MDP with N states and A actions, the number of samples necessary for the expected reward to converge within ε of the optimal policy with probability $1-p$ is bounded by:

$$O\left(NA\varepsilon^{-2}\left(\log\left(\frac{N}{p}\right) + \log\left(\log\left(\varepsilon^{-1}\right)\right)\right)\right). \tag{12}$$

For fixed values of ε and p, the number of training steps is $O(NA\log(N))$, but the constant factor can be significant. Due to the curse of dimensionality, the number of states of the MDP tends to be very large for all but the most trivial problems, making standard Q-learning need a huge amount of training samples to reach convergence and obtaining a good trade-off between precision and adaptability.

Depending on the definition of the video adaptation MDP, standard Q-learning algorithms can either be fast to converge and adaptable in an online setting [26], [27] or efficient after convergence [25]: the number of states necessary to accurately represent the environment makes Q-learning slow and unwieldy.

Our objective is to concoct RL algorithms that converge quickly and that are capable of generalizing based on experience, i.e., that can cope with *previously unseen* channel/quality patterns, and that approximate well the optimal policies that would be obtained by solving the MDP. To achieve this, referring to s' as the state of the system upon taking action q and by r the actual reward accrued from that action, we advocate

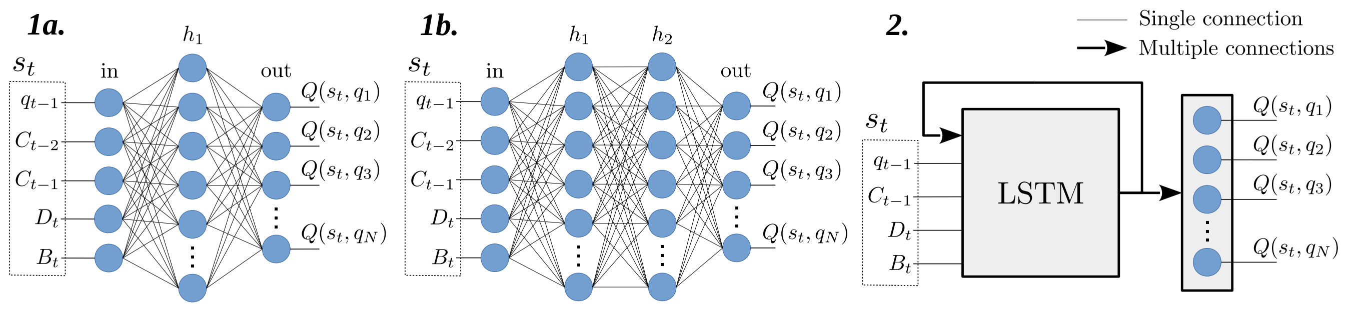

Fig. 1. Network architectures considered in this study: an MLP network with one (***1a***) and two (***1b***) hidden layers; a RNN based on an LSTM cell (***2***). For this plot, the channel memory was assumed $n = 2$, i.e., $\boldsymbol{C}_t = [C_{t-2}, C_{t-1}]$.

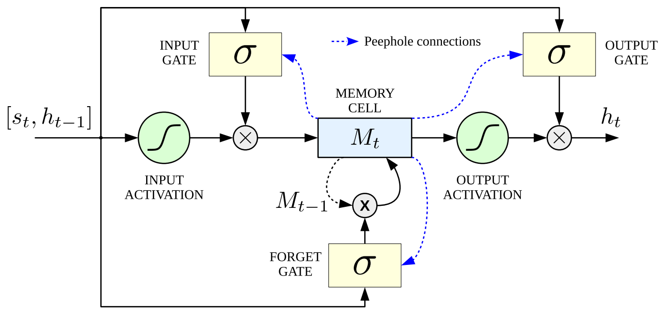

Fig. 2. Schematic diagram of an LSTM cell.

using the 4-tuple (s, q, r, s') to update $Q(s, q)$ and, along the same lines of [29], to concurrently improve the Q-values associated with other states and actions. As we explain below, in this paper we obtain this through Deep Q-learning, as it provides a natural and effective way to generalize the knowledge acquired during specific transitions and reuse it for other states and actions.

C. Deep-Learning Integration

Considering the scenario that we tackle in this paper, Q-learning has two main drawbacks:

- *Continuous state space:* The quantities defining the state space, namely, the buffer state and the channel capacity take value in some real interval. The definition of an MDP of manageable size involves a quantization of buffer and channel capacity according to a predefined number of levels (dictated by the quantization step). The smaller the quantization step, the better the approximation of the actual (continuous) variable and the more accurate the fit between the MDP and the process it represents. However, the number of states grows very quickly as the quantization step gets smaller. The best compromise between representation accuracy and number of states is often difficult to find;
- *Curse of dimensionality:* This is a direct consequence of the previous point, since the number of samples required for Q-learning to converge grows very quickly with the number of states, according to (12). Here, the problem is not only related to the convergence time, but also to the data availability: optimal policies may be hard to attain due to the need for too many data samples, which may not be available in practical settings.

With *deep Q-learning* we cope with these issues by approximating the action-value function $Q(s, q)$ through a neural network that, for any given state and action pair (s, q), returns the corresponding (estimated) Q-value $Q(s, q)$. The network weights, once trained, will encode the mapping and replace the tables used by Q-learning. This allows the model to be fed with continuous variables, avoiding the quantization problem, and has the further desirable property that neural networks, if properly trained, are able to *generalize*, providing correct answers (excellent Q-value approximations) even for points (s, q) that were never processed in the training phase. In other words, neural networks act as universal approximators. As we shall quantify below, in Section VII-D, this amounts to a reduction of the number of training samples that are required to reach a certain performance level. With this approach, the RL logic remains unchanged, and there is no restriction on the neural network architecture to use. The network's weights are updated to approximate the optimal action-value function $Q^*(s, q)$ using typical gradient descent optimization methods: the numerical results shown in this paper have been obtained by using the Adaptive Moment Estimation (Adam) method [37], in conjunction with backpropagation. The use of neural networks within the proposed architecture for video-streaming control is further discussed in Section VII-D.

V. Neural Network Architectures: Preliminaries

A typical neural network consists of many simple units, called neurons, connected in several ways which depend on the implemented architecture. Input neurons get activated directly by the environment state variables, while other neurons are activated through weighted connections from neurons residing in previous layers. Given the architecture, i.e., the way neurons are connected, and the activation functions, i.e., how the weighted input is reshaped by each neuron (and subsequently sent forward to the next layer), the whole system is completely determined by the connection weights and biases, which are both included in a single vector $\boldsymbol{w}$ in the following. Any type of neural network can be used to approximate the action-value function. In this paper, we explore two different network architectures, specifically Multilayer Perceptron (MLP) and Long-Short Term Memory (LSTM) networks.

An MLP is a fully connected feed-forward network with one or more hidden layers (see Fig. 1, subfigures (1a) and

(1b)). The output of any neuron in any layer $\ell \geq 1$ is obtained by first computing a *weighted* sum of all the outputs from the previous layer, and then evaluating it through a non-linear activation function (the hyperbolic tangent, in our case). The final output vector, from the neurons in the last layer, is obtained through a cascade of operations of this type, by passing the output vector from any layer $\ell \geq 1$ to all neurons in the next layer $\ell + 1$. This network has no memory, and this means that given an input vector, the final output vector only depends on the network's weights. In this case, if we desire to keep track of some memory, such as the temporal evolution of video samples up to a certain number of instants (video segments) in the past, the input vector has to be extended to contain this information. This entails a redefinition of the system state (to account for *current* and *past* samples), which corresponds to a larger number of neurons and to a higher complexity (in terms of training time and memory space).

On the other hand, Recurrent Neural Networks (RNNs) implement some internal feedback mechanics that introduce memory, i.e., given an input vector, the network output depends on the network's weights *and* on the previous inputs. In this study, we implement an RNN network based on LSTM cells [38] as sketched in Fig. 1(2). A schematic diagram of the LSTM internal structure is shown in Fig. 2. The memory is implemented through a Memory Cell that allows storing or forgetting information about past network states. This is made possible by structures called gates that handle access to the Memory Cell. Gates are composed of a cascade of a network with sigmoidal activation function (σ) and a point-wise multiplication block. There are three gates in an LSTM cell: **1.** the *input gate*, that controls the new information that need to be stored in the Memory Cell, **2.** the *forget gate*, that manages the information to keep in the memory and what to forget, and **3.** the *output gate*, associated with the output of the cell (h_t). In addition, all the data that pass through a gate is reshaped by an activation function (usually an hyperbolic tangent). Optionally, peephole connections can be added to allow all gates inspect the current cell state, even when the output gate is closed [39]. Backpropagation Through Time (BTT) is usually used in conjunction with optimization methods to train RNNs [40].

VI. Deep Q-Learning for DASH Adaptation

Conventional machine learning techniques are often limited in their ability to analyze data in their natural form. Usually, a good representation of the environment requires complex analysis and considerable expertise. This phase is commonly referred to as *feature engineering* and aims at finding a suitable representation of the raw data through a reduced set of features (*feature vector*), from which the learning system can extract useful environment information.

Representation learning consists of a set of mechanisms to automate this process: the learning machine is fed with raw data and discovers the best representation for detection or classification on its own. Deep-learning methods are representation learning techniques which use multiple successive layers of artificial neurons; each layer is composed of simple (but non-linear) modules that transform the input representation into a slightly more abstract one, which is then used as the input for the next layer. With a complex enough deep structure, even very complex functions can be learned successfully. A great deal of work has been carried out on deep-learning architectures in the last decade [41], [42]. For further details, see [43].

The presented D-DASH framework combines a Q-learning approach with a deep-learning framework to obtain optimal policies for the DASH protocol adaptation engine, as described in Section IV. Learning systems of this kind, referred to as *Deep Q-networks (DQNs)*, have been used in many complex systems in different research fields with state of the art performance, although their development is quite recent.

The main difference between standard Q-learning, as described in Section IV-B, and DQN, is in the way of estimating the Q-value of each state-action pair, generalized by the function $Q(s, q)$, i.e., an approximation of the optimal action-value function $Q^*(s, q)$. While standard Q-learning keeps a table of values and updates each state-value pair separately, DQN uses a deep-learning approach to approximate the Q-function. This can be achieved in two different ways:

1) a single deep network, fed with the current system state, is used to simultaneously estimate the Q-values for all possible actions;
2) one separate deep network is trained *for each* possible action, approximating a sub-space of the whole action-state set.

In this study we utilize the first approach, as it has the advantage of providing the entire set of Q-values (always needed to make the final decision) with a single computation.

Considering the MDP defined in Section IV-A, a loss function $\tilde{L}$ at iteration t is evaluated using the following 4-tuple $e_t = (s_t, q_t, r_t, s_{t+1})$, which here is referred to as the *agent's experience* at time t. The loss function can be derived from the Bellman equation in (11) as follows [44]:

$$\begin{aligned} &\tilde{L}_t(s_t, q_t, r_t, s_{t+1}|\boldsymbol{w}_t) = \\ &= \left(r_t + \lambda \max_q \hat{Q}(s_{t+1}, q|\bar{\boldsymbol{w}}_t) - Q(s_t, q_t|\boldsymbol{w}_t) \right)^2, \end{aligned} \quad (13)$$

where r_t is the reward accrued for segment t, for which we use (4). Note that the framework is rather general and any QoE function $S(q_t)$ can be plugged in without requiring any change to the model. Two deep neural networks, with the same architecture, are considered. A first network, with weight vector $\boldsymbol{w}_t$, is updated for each new segment (at every time step t), and is used to build the Q-value map $Q(s_t, q_t|\boldsymbol{w}_t)$. A second neural network, referred to as the *target network*, is accounted to increase the stability of the learning system [44], and its weight vector $\bar{\boldsymbol{w}}_t$ is updated every K steps (segments), by setting it equal to that of the first network and keeping it fixed for the next $K - 1$ steps, i.e., $\bar{\boldsymbol{w}}_t = \boldsymbol{w}_t$ every K steps. The target network is used to retrieve the mapping $\hat{Q}(s_{t+1}, q|\bar{\boldsymbol{w}}_t)$ in (13). Another improvement is given by the implementation of a technique called *experience replay* [45]. The agent's experience $e_t = (s_t, q_t, r_t, s_{t+1})$ is stored into a *replay memory* $R = \{e_1, \ldots, e_t\}$ after each iteration. In this way, a new loss function L_t that also accounts for the past experience can

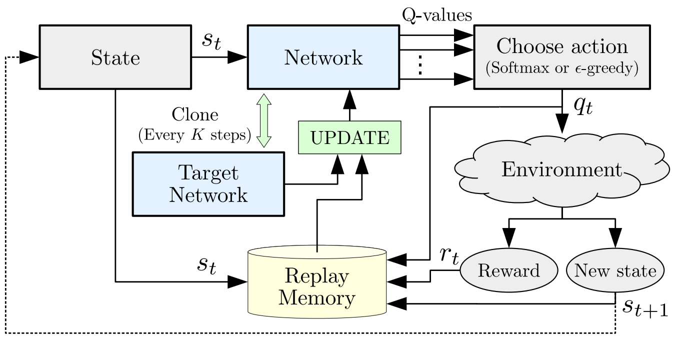

Fig. 3. Schematic diagram of an update iteration.

be considered. Specifically, a subset $R_M = \{e_1, \dots, e_M\}$ of M samples, with $e_j \in R, j = 1, 2, \dots, M$, is extracted uniformly at random from the replay memory R, and L_t is finally evaluated as an empirical mean over the samples in set R_M:

$$L_t(\mathbf{w}_t) = \frac{1}{M} \sum_{e_j \in R_M} \tilde{L}_t\big(e_j|\mathbf{w}_t\big). \tag{14}$$

This leads to three main advantages: greater data efficiency, uncorrelated subsequent training samples and independence between current policy and samples [44].

The whole process can be divided into two consecutive phases, which take a different but fixed number of iterations: namely, the *update phase*, and the *test phase*.

Update phase: The exploration parameter, namely ε in the case of an ε-greedy policy or ξ for softmax (see Section IV-B), is gradually reduced. We recall that a smaller value for this parameter means that the policy tends to prefer the action that is considered to be optimal at the current training stage. Furthermore, at each iteration the network's weights are updated to minimize the loss function in (14). The Adam method is used as the gradient descent optimization algorithm: it implements an adaptive learning rate to provide a faster and more robust convergence [37].

Test phase: The exploration parameter is set to zero, thus obtaining a *greedy policy* implementing the actions that are deemed optimal given the current system state and the mapping $Q(s_t, q_t|\mathbf{w}_t)$ from the first neural network. For this phase, the weights $\mathbf{w}_t$ are frozen and are no longer updated for the whole duration of the test. The target network is not used in the test phase and all the performance evaluations are based on the results obtained during this second phase.

A schematic diagram of an update iteration is shown in Fig. 3. First, the current environment state s_t is fed to the deep neural network, which outputs an estimate of the Q-value for each possible action $q \in A$, i.e., the various representations in the adaptation set A. Then, an action q_t is chosen according to either an ε-greedy or softmax policy. Upon taking action q_t, the system moves to the new state s_{t+1} and a new reward r_t is evaluated according to (4). The newly acquired experience $e_t = (s_t, q_t, r_t, s_{t+1})$ is stored into the replay memory R. Hence, a batch of M samples is extracted, uniformly at random, from the replay memory and is used to update the network's weights through the Adam optimization method. The loss function in (14) is minimized, using the mapping

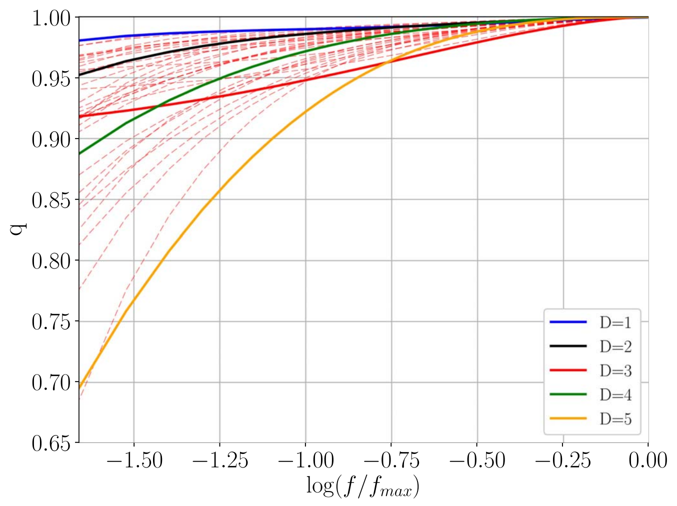

Fig. 4. Reference quality-rate curves.

from the target network, i.e., $\hat{Q}(s_{t+1}, q|\bar{\mathbf{w}}_t)$, whose weights $\bar{\mathbf{w}}_t$ are updated every K steps.

Two different types of deep network architectures have been tested in this study (see Fig. 1): **1.** a fully connected feed-forward network, and in particular an MLP network with one (Figure *1a*) and two (Figure *1b*) hidden layers, respectively referred to as MLP$_1$ and MLP$_2$ in the following, and **2.** a recurrent neural network based on a LSTM cell [38]. An LSTM with deep hidden-to-output function has been implemented [46]. Accordingly, the LSTM output is processed by another fully connected layer to provide the final Q-values. We remark that the number of previous components considered in the capacity vector $\boldsymbol{C}_t$ of the input state differs in the two cases: feed-forward networks (MLP$_1$ and MLP$_2$) require to be fed with the whole vector $\boldsymbol{C}_t$, whereas recurrent networks only need the last channel capacity sample C_{t-1} as input, as the feedback loop inside the LSTM cell keeps track of the channel memory. The dimension of vector $\boldsymbol{C}_t$ has to be fixed beforehand for the feed-forward networks, and cannot be adapted after the definition of the network. In this paper, we consider 2 history samples for MLP$_1$ and MLP$_2$, and 5 history samples for a *"long history"* (lh) version of MLP$_2$, referred to as MLP$_{\text{lh}}$.

The LSTM has the advantage of automatically learning what to store inside the memory cell, and what to forget. Here, in addition to standard LSTM, we also consider an LSTM cell with peephole connections, referred to as LSTM$_{\text{ph}}$.

Due to the continuous changes of the target function and training data, we do not expect serious overfitting issues. However, a dropout regularization technique has been applied to MLP$_2$ (termed MLP$_{\text{do}}$) to verify this assumption. Dropout is a well-known and simple method to prevent neural networks from overfitting [47]. It amounts to randomly dropping neurons and their connections, with a certain probability. A 20% dropout probability is considered in the training of MLP$_{\text{do}}$.

VII. Simulation and Results

In this work, we assume segments belonging to the same representation set have a constant size and variable quality;

TABLE I
SIMULATION PARAMETERS

PARAMETER	VALUES
T	2 s
$f_{\max}$	20 Mb
$F_t(\cdot)/T$	{0.25, 0.5, 1, 2, 3, 4, 6, 10} Mb/s
C_t (pretrain)	{0.4, 0.75, 1.5, 2.5, 3.5, 4.5, 5.75, 7.25, 9, 12.5} Mb/s
$D_t = 1$ (Akiyo)	$\mathbf{d}_t = [0.99947, -0.01015, -0.02888, -0.02427, 0.00415]$
$D_t = 2$ (News)	$\mathbf{d}_t = [0.99970, -0.01064, -0.02291, -0.02531, 0.00074]$
$D_t = 3$ (Bridge - far)	$\mathbf{d}_t = [1.00033, -0.01051, -0.05385, -0.08211, 0.01361]$
$D_t = 4$ (Harbor)	$\mathbf{d}_t = [0.99977, -0.00505, 0.00554, -0.01726, 0.00022]$
$D_t = 5$ (Husky)	$\mathbf{d}_t = [0.99984, 0.00998, 0.07590, -0.01138, 0.00040]$

this is a common assumption in the relevant literature. In real systems, the encoding parameters of the video are often constant, with segments of varying size [48]; however, the performance of the learning algorithms should not be significantly affected by this factor, and the model presented in Section III is fully general.

To evaluate the performance of the proposed algorithms, we carried out extensive trace-based simulations, where the different components of the system have been modeled in a realistic manner from real data traces. More specifically, the video model was derived from real videos from the EvalVid database.[1] The video traces were characterized in terms of their quality-rate function where, as previously explained, we used SSIM [12] as the instantaneous video quality metric. Then, we derived the SSIM-rate characteristics of a large number of videos in the dataset, from which we extracted a limited number of reference SSIM-rate curves. Following [49], such curves have been represented as 4th-degree polynomials function of the bitrate, so that the quality (SSIM) of a given video sequence encoded at rate $f/f_{max} \in (0, 1]$ is given by

$$q \simeq d_0 + \sum_{k=1}^{4} d_k (\log(f/f_{\max}))^k, \tag{15}$$

where $f_{\max}$ is the full-quality video segment size, and $f \leq f_{\max}$ is the actual segment size. The vector $\mathbf{d} = [d_0, \ldots, d_4]$ offers then a synthetic representation of the complexity of a video scene. The chosen reference curves are shown in Fig. 4, and the values of each reference curve's vector $\mathbf{d}$ are listed in Table I. These reference curves were combined into scenes with exponentially distributed duration, a model validated by Rose [33], with an average of 10 seconds (i.e., 5 segments). In this way, we generated a large number of realistic video traces for our tests. In our numerical results, we picked 8 different values for the segment size f: if q is the segment quality, f is defined as $f = F_t(q)$, where $F_t(\cdot)$ is the inverse of (15), see also Table II for the considered adaptation set. In the learning system, each video segment is characterized by a vector $\mathbf{d}_t$, which is summarized by an index D_t, as shown in Table II.

To model the transport capacity of the HTTP connection we considered three datasets: a) *real* traces of HTTP video streaming sessions over LTE [50]; b) *synthetic* traces obtained through the network simulation platform `ns3`; c) *Markovian* traces obtained by means of a simple Markov model. The learning algorithms were first trained on the

[1]http://www2.tkn.tu-berlin.de/research/evalvid/cif.html

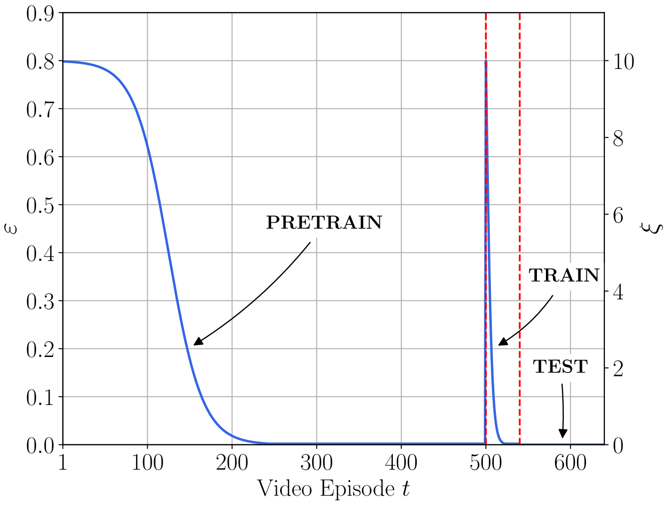

Fig. 5. Profile of the exploration rate during the training and testing phases.

TABLE II
ADAPTATION ALGORITHM PARAMETERS

ALGORITHM	PARAMETER	VALUE
FESTIVE	target buffer	15 s
	buffer randomness	0.25 s
	α	10
	k_S (switch memory)	10
	k_C (throughput memory)	5
MPC	maximum buffer size	30 s
	γ	2
	μ	50
	K	5
All learners	β	2
	maximum buffer size	20 s
	γ	50
	δ	0.001
	B_{thr} (safe buffer)	10 s
Q-learning	α	0.1
	policy	Softmax
	q_t (SSIM)	{0.84, 0.87, 0.9, 0.92, 0.94, 0.96, 0.98, 0.99, 0.995}
	C_t (Mb/s)	{0.5, 1, 2, 3, 4, 5, 6, 8, 10}
MLP_1	hidden neurons N_h	256
	learning rate	10^{-3}
	batchsize M	1000
	K	20
	policy	Softmax
	λ	0.9
MLP_2, MLP_{do}, MLP_{lh}	hidden neurons N_{h1}, N_{h2}	128, 256
	learning rate	10^{-4}
	batchsize M	1000
	K	20
	policy	ε-greedy
	λ	0.9
LSTM, LSTM_{ph}	number of units N_c	128
	learning rate	10^{-3}
	batchsize M	100
	K	200
	policy	Softmax
	λ	0.5

Markovian channel model (*pretrain* phase), then shown a small number of realistic traces (either *real* or *synthetic*) as an actual training (*train* phase). The neural networks' weights

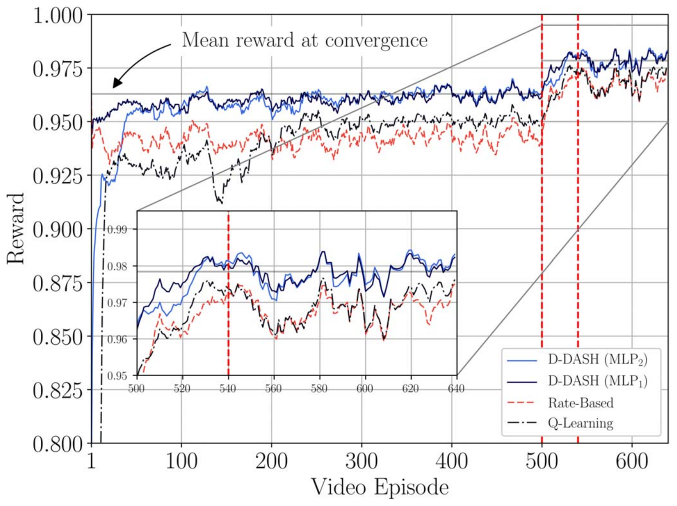

Fig. 6. Reward for the standard Q-learning, D-DASH and rate-based algorithm (using real traces as test set). The pretrain phase takes the first 500 video episodes, the training phase is enclosed within the two vertical lines and the test phase takes the remaining points, after the second line.

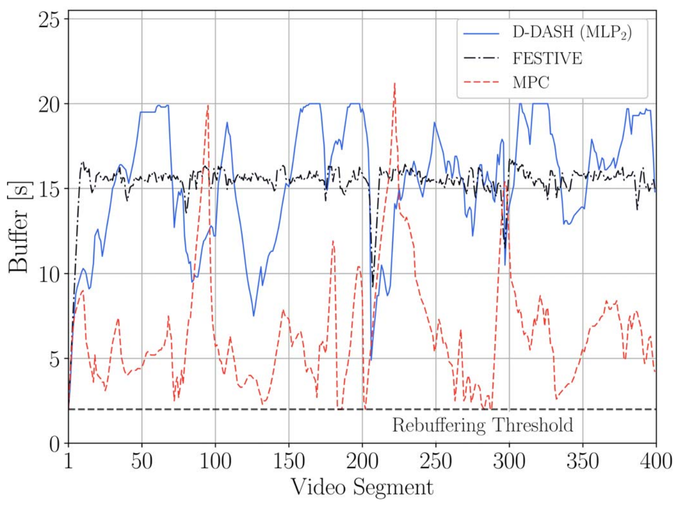

Fig. 7. Evolution of the buffer during a video episode in the *test* phase.

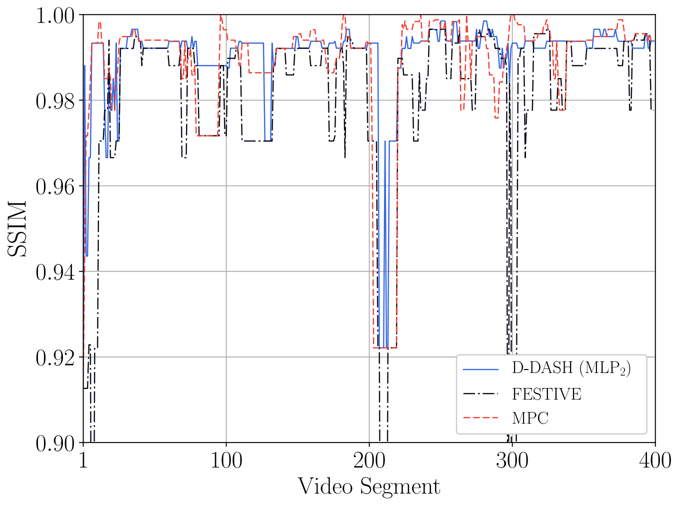

Fig. 8. Evolution of the SSIM during a video episode in the *test* phase.

were then frozen and their exploration parameter was set to zero to carry out a fair comparison against state of the art algorithms from the literature using other traces from the same dataset (*test* phase). The *pretrain* phase lasted for 500 synthetic videos, while the *train* phase only lasted for 40 actual videos. The final *test* phase was performed over 100 actual video episodes; each video episode in all phases lasted 400 segments of $T = 2$ s each. The purpose of the *pretrain* phase is to teach the learning agent the basic mechanics of video streaming in a highly variable and realistic network scenario. The rationale is that the solution so obtained should represent a sensible starting point for the learning process when confronted with real traces, leading to a much shorter training time.

The Markov model for the *pretrain* phase used the same settings as in [29]; the capacity evolution was controlled by a random walk model among a set of levels, with a state change probability of 0.5. The possible capacity levels C_t are listed in Table I. We remark that the Markovian traces were only used in the pre-training phase, while the performance evaluation was carried out by considering realistic traces, either from real HTTP measurements or generated by ns3.

Each component of the reward function in (4) was scaled into the range between 0 and 1 to avoid numerical convergence issues.

A. Algorithm Settings

The algorithms from the literature selected as benchmarks were the parallel Q-learning and the simple rate-based heuristic from [29], MPC [20], which is a dynamic programming-like solution, and FESTIVE [17], which is among the most conservative heuristic-based algorithms, thus generally guaranteeing good stability and a low number of rebuffering events. One of FESTIVE's aims is providing a stable quality for multiple clients, which is beyond the scope of this work, so its performance should be evaluated accordingly. Another point that bears consideration is the computational complexity of MPC: the authors themselves state that a real-time implementation would require pre-computed tables, which leads to memory occupation. We also implemented the PANDA [18] and QoE-RAHAS [19] algorithms, but their performance was significantly worse than all the other algorithms' for any configuration of their parameters in the simulation environment, so we did not show them in any of the plots.

The standard Q-learning algorithm used a Softmax policy; the other learning algorithms were tested with both policies, and the best-performing one was chosen in each case. The policy for each algorithm is listed in Table II.

The hyperparameters of the learning algorithms were optimized by performing an exhaustive search and selecting the combinations that performed best in the *pretrain* phase; the values of the parameters for all algorithms are listed in Table II, except for the exploration parameter (either ε or ξ, depending on the exploration mode), which followed the profile shown in Fig. 5 over the three phases. The settings

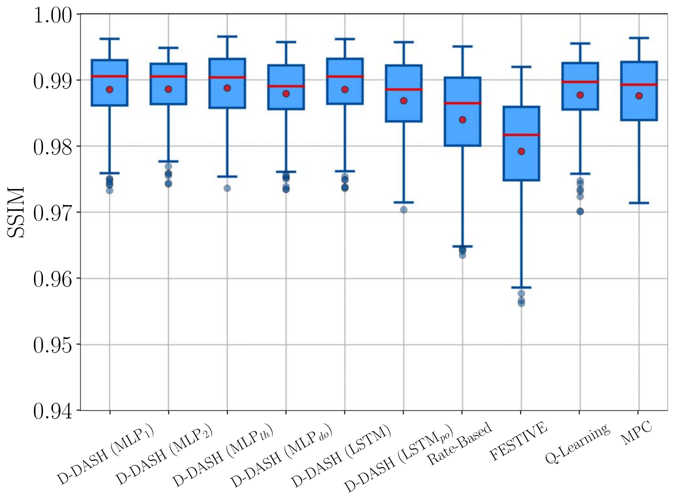

Fig. 9. Boxplot of the average SSIM for the 100 video episodes in the *test* phase (real traces).

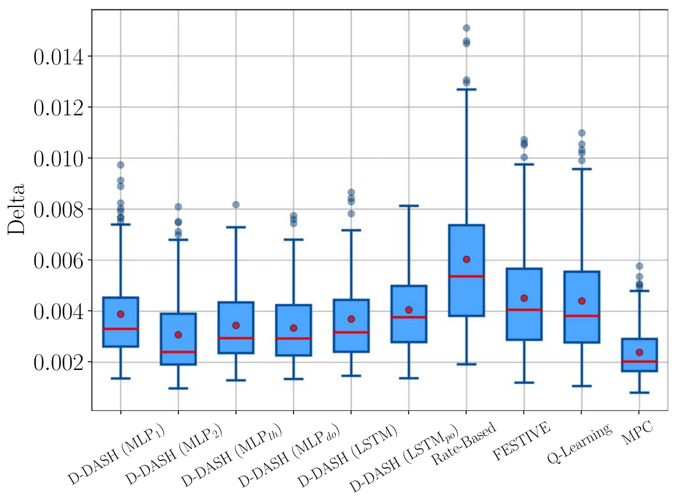

Fig. 10. Boxplot of the average SSIM variation for the 100 video episodes in the *test* phase (real traces).

and model for FESTIVE and the standard Q-learning algorithm are the same used in [17] and [29] respectively, except where otherwise stated. In Fig. 5, each video episode has $M = 400$ segments, the standard Q-learning algorithm uses preset thresholds to quantize the video quality and capacity measurements, which are listed in Table II.

We adapted the MPC parameters to reflect our definition of QoE and provide a fair comparison, as its basic model is very similar to ours and using the same QoE function provides a more meaningful comparison.

B. Results: Real Traces

Our simulations aimed to assess the performance of the D-DASH framework with regard to the three major video quality metrics, i.e., instantaneous quality, its stability and the frequency of rebuffering events. We compared D-DASH based algorithms against existing approaches from the literature, while also investigating their convergence speed.

Fig. 6 shows a smoothed version of the reward over the three phases of the simulation for the MLP_1 and MLP_2, using standard Q-learning and the rate-based heuristic as a comparison. The tests were performed by freezing the learner state and setting a greedy policy (i.e., always taking the action with the highest expected reward) after each video episode. The benefits of the Deep-Q approach appear evident at a first glance; standard Q-learning gets a lower reward at convergence, as well as taking more than 200 episodes to overtake the rate-based heuristic and failing to adapt quickly enough to the real traces in the *train* phase.

The two D-DASH algorithms achieve higher rewards after just a few videos: MLP_2 converges in the first 50 videos, while MLP_1 already reaches its peak performance after the first video episode. The same trend can be seen in the *train* phase, where both schemes maintain a significant advantage on the simple heuristic and standard Q-learning.

One of the main advantages of smart QoE-aware adaptation schemes is a better buffer management: while most classical algorithms try to keep the buffer as stable as possible, the

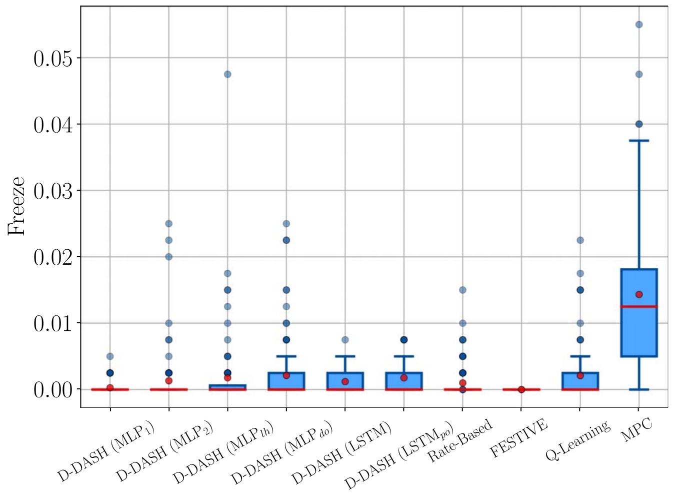

Fig. 11. Boxplot of the frequency of rebuffering events for the 100 video episodes in the *test* phase (real traces).

D-DASH algorithms can use the buffered segments when the available capacity drops and build up the buffer when it is convenient to do so. Fig. 7 shows an example taken from the *test* phase: while FESTIVE keeps an extremely stable buffer (except for a sudden drop after about 200 segments, due to a sharp change in the capacity), the MLP_2 algorithm has a large dynamic range, building up the buffer when capacity is high and using it up to cover temporary outages. The other D-DASH algorithms make a similar use of the buffer, as well as standard Q-learning. MPC also maintains a high and stable quality throughout the video, but it incurs in several rebuffering events because of its optimistic throughput prediction, as Fig. 7 shows.

The benefits of the smarter buffer use are clearly visible in the quality graph of Fig. 8: MLP_2 can avoid sharp drops in the video quality without triggering rebuffering events.

In the next figures we compare the performance of the different algorithms in the *test* phase, in terms of image quality (SSIM), rebuffering, and quality stability.

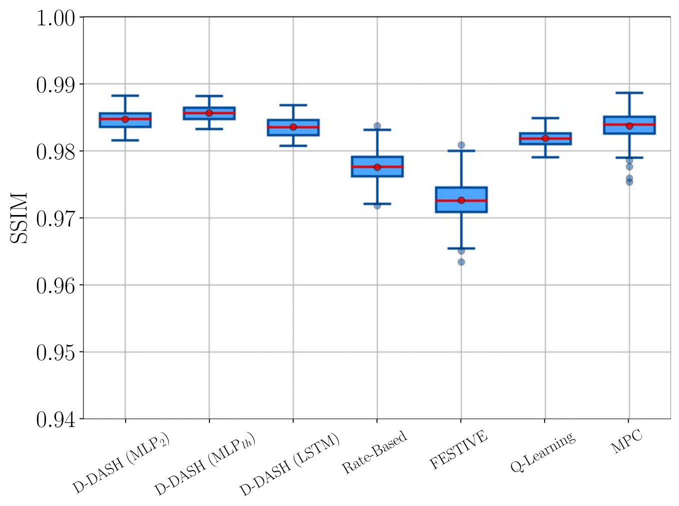

Fig. 12. Boxplot of the average SSIM for the 100 video episodes in the *test* phase (synthetic traces).

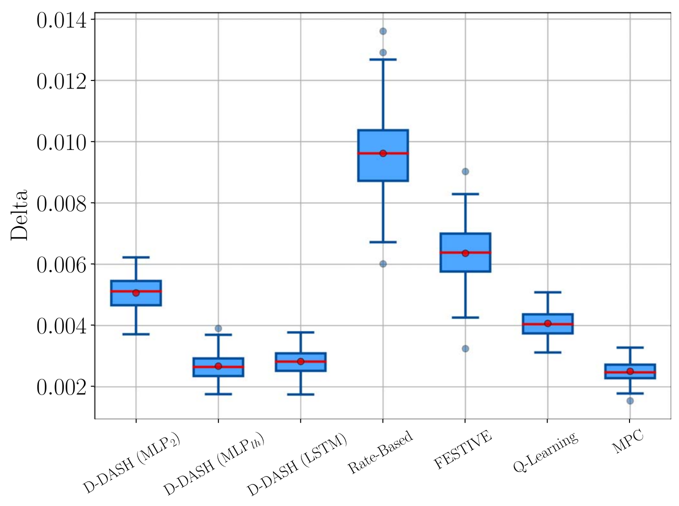

Fig. 13. Boxplot of the average SSIM variation for the 100 video episodes in the *test* phase (synthetic traces).

Fig. 9 shows the achieved SSIM: all the learning solutions, including standard Q-learning, achieve a higher SSIM than FESTIVE and the rate-based heuristic; aside from the higher median, the bottom 5% of the videos still have an average SSIM of over 0.97, while FESTIVE and the rate-based heuristic go below that threshold for a significant fraction of the videos. It should be noted that FESTIVE has a lower average SSIM than the rate-based heuristic, since it privileges stability over instantaneous quality. MPC performs about as well as the learning-based algorithms, but with a slightly larger variance.

Fig. 10 shows the average difference between the SSIM of one segment and the next. As expected, FESTIVE keeps the quality more stable than the rate-based heuristic. Similar performance is obtained by standard Q-learning, which also keeps a higher average SSIM. The two Deep-Q algorithms outperform FESTIVE and Q-learning, but the best quality stability is achieved by MPC, with an average SSIM variation always lower than 0.006, while that of the other algorithms reaches 0.008 for at least one video.

Finally, Fig. 11 shows the frequency of rebuffering events for each algorithm: since FESTIVE is extremely conservative, there were no rebuffering events over the whole *test* phase. However, even the most aggressive learning algorithms (standard Q-learning and MLP$_{do}$) do not experience rebuffering events often. The LSTM and LSTM$_{ph}$ algorithms experience at least one rebuffering in 25% of the videos, but they never have more than three, and one rebuffering over a whole 400 segment video episode can be considered acceptable. MPC pays for its higher stability by having an average of 5 rebuffering events per video, far higher than any of the other algorithms'. MPC's reliance on an imperfect throughput predictor shows its limits when dealing with highly variable environments.

We also included another version of MLP$_2$, called MLP$_{lh}$, which uses the last 5 throughput samples as input instead of the last 2. Its aim is to show the benefits of a longer memory in the presence of long-term correlation. We observe that, with real channel capacity traces, the performance of MLP$_{lh}$ are basically the same of LSTM, probably because

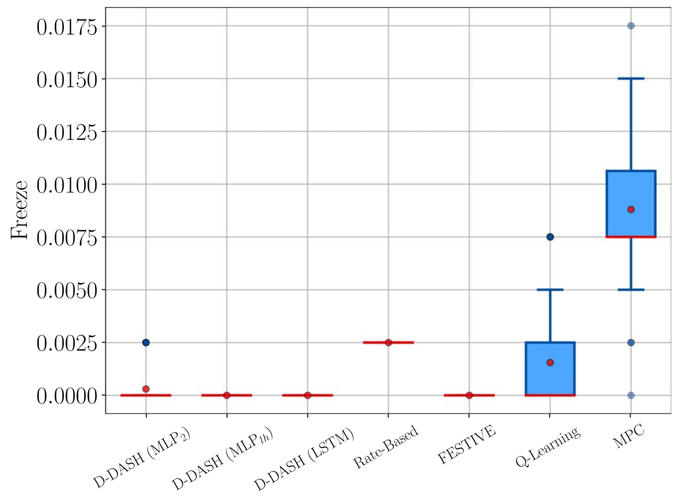

Fig. 14. Boxplot of the frequency of rebuffering events for the 100 video episodes in the *test* phase (synthetic traces).

the correlation of the empirical channel data considered in this study is low, so that the extra memory in the LSTM is not necessary. Finally, the use of dropout techniques (MLP$_{do}$) to avoid overfitting does not provide any improvement, as expected.

Its relative simplicity, low rebuffering rate and quick convergence arguably make MLP$_1$ the best adaptation algorithm in this scenario: aside from some very rare rebuffering events, it is an improvement over the state of the art in all the considered QoE metrics over real capacity traces. Nevertheless, as shown in the following section, any network with a limited memory shows its limitations when a more complicated scenario is analyzed, and the presence of long-term correlation in the channel capacity makes a more complex approach necessary.

C. Results: Synthetic Traces

After running the adaptation algorithms over real capacity traces, we generated a set of traces with the *ns-3* simulator to gauge the effects of TCP cross-traffic on the algorithms' performance. The traces were generated by measuring the throughput of a saturated TCP flow sharing a bottleneck with a capacity of 10 Mb/s and a latency of 50 ms with 19 other TCP clients. Each of the cross-traffic clients generated TCP traffic as an on/off source: the off period was exponentially distributed with a mean of 2 seconds, while the on time was set to 4 seconds. This traffic model was designed to introduce long-term correlations in the available capacity [51], which are notoriously hard to handle for adaptation algorithms. In this part of the study, we only tested the MLP$_2$ (both with the short and long memory) and LSTM algorithms for clarity, since the performance of the other variants was similar.

Fig. 12 shows the average SSIM over the *test* phase using the synthetic traces. In this situation, the advantage of the D-DASH algorithms is more marked: the MLP$_2$ and LSTM are able to maintain an average SSIM above 0.98 for all the considered videos; standard Q-learning and MPC also perform better than the Rate-Based heuristic and FESTIVE.

Fig. 13 shows the net advantage of a long memory when dealing with long-term correlations: although the average SSIM of LSTM and MLP$_{\text{lh}}$ is the same as that of MLP$_2$ they can achieve it with half of the quality variation. Even if MLP$_{\text{lh}}$ performs slightly better than LSTM, it considers a fixed amount of memory. The LSTM network has the additional capability to automatically adapt to different memory requirements, without increasing the state space dimension. The D-DASH algorithms and standard Q-learning all perform significantly better than FESTIVE and the Rate-Based heuristic on this metric. As with the real traces, MPC is the best at keeping a stable quality among the considered algorithms.

Fig. 14 shows the real advantage of the D-DASH framework in this scenario: while standard Q-learning has almost the same average SSIM as LSTM, MLP$_{\text{lh}}$ and MLP$_2$, and actually has smaller quality fluctuations than the latter, it can only achieve this at the cost of relatively frequent rebufferings: a quarter of the video episodes have at least one rebuffering event, while this only happens for a few episodes with MLP$_2$ and never for LSTM. FESTIVE is also able to avoid rebuffering events, while the Rate-based heuristic is not. MPC still suffers from a high number of rebuffering events, totaling an average of 3 events per video episode.

Fig. 15 shows the higher stability of LSTM with respect to MLP$_2$: for the sake of a higher stability LSTM avoids some of the quality increases. For this complex channel, LSTM is able to predict more accurately when a certain quality increase is likely to be sustainable, i.e., without having to shortly move back to the previous quality setting. Finally, a comparison of the convergence speed between standard Q-learning and D-DASH, performed during the pretrain phase, is provided in Fig. 16: D-DASH algorithms converge much faster, making a better use of the video examples that are supplied during the learning phase.

D. Memory Allocation

Since adaptation algorithms are client-side, their memory footprint is an important consideration. The number of variables required for D-DASH, and also for the classic Q-learning (N_Q), are as follows:

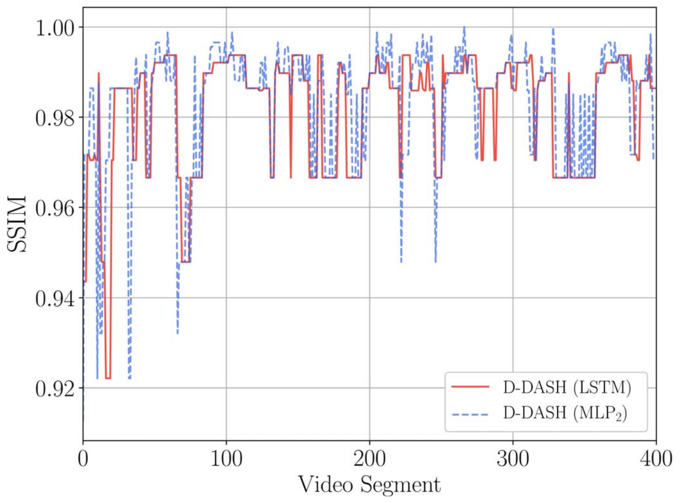

Fig. 15. Evolution of the SSIM during a video episode in the *test* phase, using the synthetic traces.

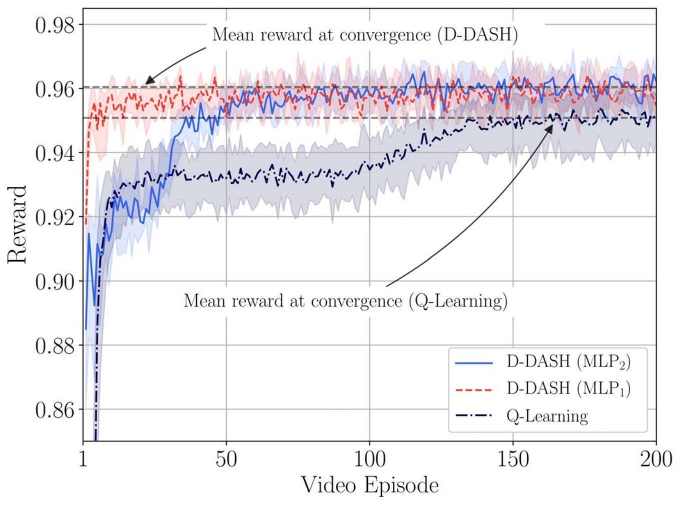

Fig. 16. Comparison of the convergence speed between standard Q-learning and D-DASH implementations (MLP$_1$ and MLP$_2$). The shaded area represents the interquartile range.

$$N_Q = N_s N_a \tag{16}$$
$$N_{MLP_1} = (V_s + 1)N_h + (N_h + 1)N_a \tag{17}$$
$$N_{MLP_2} = (V_s + 1)N_{h1} + (N_{h1} + 1)N_{h2} + (N_{h2} + 1)N_a \tag{18}$$
$$N_{LSTM} = 4(V_s + N_c + 1)N_c + (N_c + 1)N_a \tag{19}$$

where N_s is the cardinality of the state set, V_s is the number of state variables and N_a is the cardinality of the action set. Note that V_s and N_a also corresponds to the number of inputs and outputs of the neural networks, respectively (see also Fig. 1). N_h, N_{h1} and N_{h2} are the number of hidden layer's neurons in the MLP networks, and N_c are the number of units in the LSTM cell. According to (17), (18) and (19), considering the values in Table II and 32 bit floating-point representation for storing real variables, the memory space required for MLP$_1$, MLP$_2$ and LSTM is about 14.4 kB, 143.4 kB and

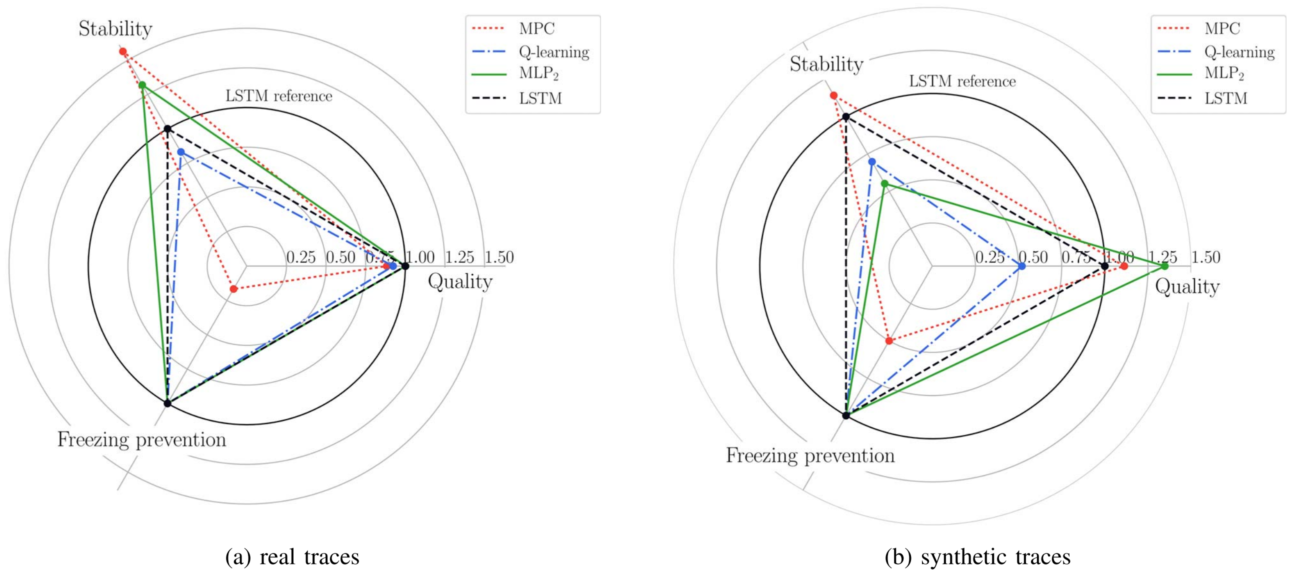

Fig. 17. Summary of performance of video adaptation algorithms: (a) real traces, (b) synthetic traces (exhibit long-term correlation).

276.5 kB, respectively. We remark that these values are fixed given a specific network implementation, in the sense that they do not depend on the number of states. On the other hand, classic Q-learning space requirement directly depends on the cardinality of the state set, which is closely related to the quantization granularity of continuous state variables, as discussed in Section IV-C. The granularity that was used for the results in this paper leads to a total memory space of 32 kB for Q-learning. The memory footprint needed by each of the presented methods appears reasonable considering the hardware of modern client devices. Note that, even if the number of variables for Q-learning is lower than those required by the D-DASH algorithms, their generalization capabilities and the concurrent update of the network's weights allow for a more efficient utilization of the experience acquired in the learning phase. This behavior can be seen in Fig. 16, where the convergence speed of D-DASH is considerably lower.

E. Summary of Performance

A summary of the performance of video adaptation techniques is shown in Fig. 17. For a convenient visual comparison, the three metrics have been scaled and normalized with respect to the LSTM performance, according to the following criteria (the normalization term has been omitted for simplicity):

$$\text{Quality} = 0.98 - \text{SSIM} \tag{20}$$

$$\text{Stability} = 1/\text{Quality Variation} \tag{21}$$

$$\text{Freezing Prevention} = 0.015 - \text{Frequency of rebuffering} \tag{22}$$

The proposed Deep-Q learning based schemes significantly outperform existing Q-learning and standard techniques from the literature. As Fig. 17a shows, with real traces MPC achieves a more stable, albeit slightly lower, quality than either MLP$_2$ or LSTM, but it fails to avoid rebuffering events and results in a far worse QoE for the user. Other state of the art algorithms, such as FESTIVE, which is not shown in the plot for readability, manage to avoid rebuffering events but perform much worse in the other two metrics. Our algorithms are the only ones reaching high scores on all the three considered metrics, i.e., video quality, stability and rebuffering avoidance. Furthermore, the D-DASH algorithms converge faster compared to standard Q-learning schemes, which require hundreds of video episodes to reach an acceptable performance. In fact, they approach optimal policies after just a few videos, or even just a couple in the case of the MLP$_1$ scheme. This also makes it possible to consider an online version that adapts to video and network conditions, learning how to deal with each new situation as it arises and memorizing it for future use. The longer memory of the LSTM algorithm proved to be very valuable on channel traces with long-term correlation: when the channel correlation stretches to over 10 seconds, LSTM shows significantly better performance than other schemes in all the three metrics, as Fig. 17b shows. Also in this case, MPC obtains a higher stability than LSTM, but it pays for it by having several rebuffering events per episode.

VIII. Conclusion

In this work, we designed several Reinforcement Learning-based DASH adaptation algorithms, exploiting Deep Q-networks to speed up the convergence and adaptability of the system, and improve its efficiency. Our D-DASH framework uses different deep learning structures to approximate the Q-values, taking the raw system state as input and requiring no arbitrary design choices that might influence its performance. The deep learning algorithms also have low memory and computational requirements after an initial training phase (referred to as *pretraining* in the paper), and although the training may be computationally heavy for a mobile terminal, it can be performed offline (once for all) on a dedicated server and then passed on to the mobile device with minimal signaling overhead.

Our work represents a significant improvement on the state of the art on Q-learning DASH adaptation designs, achieving good trade-offs between policy optimality and convergence speed. D-DASH performed better than several of the most popular adaptation approaches from the literature, maintaining a high video quality without paying a significant cost either in terms of rebuffering events or stability (i.e., avoiding sudden drops in the instantaneous quality). There are several ways in which this work could be extended and built upon: aside from changes and tweaks to include more complex video content models into the algorithm input, the main challenge we foresee for intelligent DASH adaptation is collaboration with network protocols to gain additional context information. Several works in the past few years [52]–[54] have tried to use cooperation between DASH clients and network elements to improve the fairness and efficiency of video streaming on a network scale.

REFERENCES

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2015–2020 white paper," Cisco, San Jose, CA, USA, White Paper, 2015.

[2] *Dynamic Adaptive Streaming Over HTTP (DASH)—Part 1: Media Presentation Description and Segment Formats*, ISO/IEC Standard 23009-1:2014, May 2014.

[3] R. K. P. Mok, E. W. W. Chan, and R. K. C. Chang, "Measuring the quality of experience of HTTP video streaming," in *Proc. 12th IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, Dublin, Ireland, May 2011, pp. 485–492.

[4] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner, "Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming," in *Proc. 6th Int. Workshop Qual. Multimedia Exp. (QoMEX)*, Singapore, Sep. 2014, pp. 111–116.

[5] T. Hoßfeld *et al.*, "Initial delay vs. interruptions: Between the devil and the deep blue sea," in *Proc. 4th Int. Workshop Qual. Multimedia Experience (QoMEX)*, Melbourne, VIC, Australia, Jul. 2012, pp. 1–6.

[6] Z. Li *et al.*, "Streaming video over HTTP with consistent quality," in *Proc. 5th ACM Multimedia Syst. Conf. (MMSys)*, Singapore, Mar. 2014, pp. 248–258.

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

[8] A. Balachandran *et al.*, "Developing a predictive model of quality of experience for Internet video," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 339–350, 2013.

[9] P. Juluri, V. Tamarapalli, and D. Medhi, "Measurement of quality of experience of video-on-demand services: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 401–418, 1st Quart., 2016.

[10] M. Seufert *et al.*, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 469–492, 1st Quart., 2015.

[11] M. Shahid, A. Rossholm, B. Lövström, and H.-J. Zepernick, "No-reference image and video quality assessment: A classification and review of recent approaches," *EURASIP J. Image Video Process.*, vol. 2014, no. 1, pp. 1–32, Aug. 2014.

[12] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[13] C. Kreuzberger, B. Rainer, H. Hellwagner, L. Toni, and P. Frossard, "A comparative study of DASH representation sets using real user characteristics," in *Proc. 26th ACM Int. Workshop Netw. Oper. Syst. Support Digit. Audio Video*, Klagenfurt, Austria, 2016, p. 4.

[14] S. Cicalo, N. Changuel, R. Miller, B. Sayadi, and V. Tralli, "Quality-fair HTTP adaptive streaming over LTE network," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Florence, Italy, May 2014, pp. 714–718.

[15] J. De Vriendt, D. De Vleeschauwer, and D. Robinson, "Model for estimating QoE of video delivered using HTTP adaptive streaming," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, Ghent, Belgium, May 2013, pp. 1288–1293.

[16] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1842–1866, 3rd Quart., 2017.

[17] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, Nice, France, Dec. 2012, pp. 97–108.

[18] Z. Li *et al.*, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 4, pp. 719–733, Apr. 2014.

[19] S. Petrangeli and F. De Turck, "QoE-centric management of advanced multimedia services," in *Proc. IFIP Int. Conf. Auton. Infrastruct. Manag. Security*, Ghent, Belgium, Jun. 2015, pp. 50–55.

[20] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 325–338, 2015.

[21] A. Bokani, M. Hassan, and S. Kanhere, "HTTP-based adaptive streaming for mobile clients using Markov decision process," in *Proc. 20th Int. Packet Video Workshop*, San Jose, CA, USA, Dec. 2013, pp. 1–8.

[22] C. Zhou, C.-W. Lin, and Z. Guo, "mDASH: A Markov decision-based rate adaptation approach for dynamic HTTP streaming," *IEEE Trans. Multimedia*, vol. 18, no. 4, pp. 738–751, Apr. 2016.

[23] J. Lee and S. Bahk, "On the MDP-based cost minimization for video-on-demand services in a heterogeneous wireless network with multihomed terminals," *IEEE Trans. Mobile Comput.*, vol. 12, no. 9, pp. 1737–1749, Sep. 2013.

[24] S. Colonnese, F. Cuomo, T. Melodia, and R. Guida, "Cloud-assisted buffer management for HTTP-based mobile video streaming," in *Proc. 10th ACM Symp. Perform. Eval. Wireless Ad Hoc Sensor Ubiquitous Netw.*, Barcelona, Spain, Nov. 2013, pp. 1–8.

[25] M. Claeys *et al.*, "Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming," in *Proc. Adapt. Learn. Agents Workshop (ALA)*, St. Paul, MN, USA, May 2013, pp. 30–37.

[26] M. Claeys *et al.*, "Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client," *Connection Sci.*, vol. 26, no. 1, pp. 25–43, 2014.

[27] V. Martín, J. Cabrera, and N. García, "Q-learning based control algorithm for HTTP adaptive streaming," in *Proc. Int. Conf. Vis. Commun. Image Process. (VCIP)*, Singapore, Dec. 2015, pp. 1–4.

[28] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, Ottawa, ON, Canada, May 2015, pp. 131–138.

[29] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard, "Online learning adaptation strategy for DASH clients," in *Proc. 7th Int. Conf. Multimedia Syst. (MMSys)*, Klagenfurt, Austria, May 2016, p. 8.

[30] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam, "Objective video quality assessment methods: A classification, review, and performance comparison," *IEEE Trans. Broadcast.*, vol. 57, no. 2, pp. 165–182, Jun. 2011.

[31] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Trans. Image Process.*, vol. 15, no. 11, pp. 3440–3451, Nov. 2006.

[32] M. Zorzi, A. Zanella, A. Testolin, M. De Filippo De Grazia, and M. Zorzi, "Cognition-based networks: A new perspective on network optimization using learning and distributed intelligence," *IEEE Access*, vol. 3, pp. 1512–1530, 2015.

[33] O. Rose, "Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems," in *Proc. 20th Conf. Local Comput. Netw.*, Minneapolis, MN, USA, Oct. 1995, pp. 397–406.

[34] R. Bellman, "A Markovian decision process," *Indiana Univ. Math. J.*, vol. 6, no. 4, pp. 679–684, 1957.

[35] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[36] M. Kearns and S. Singh, "Finite-sample convergence rates for Q-learning and indirect algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 11. 1999, pp. 996–1002.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, San Diego, CA, USA, May 2015.

[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[39] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Mar. 2003.

[40] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[42] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[44] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[45] L.-J. Lin, "Reinforcement learning for robots using neural networks," DTIC, Fort Belvoir, AV, USA, Tech. Rep. CMU-CS-93-103, 1993.

[46] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *Proc. Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, South Brisbane, QLD, Australia, Apr. 2015, pp. 4520–4524.

[47] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[48] P. Juluri, V. Tamarapalli, and D. Medhi, "QoE management in DASH systems using the segment aware rate adaptation algorithm," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, Istanbul, Turkey, 2016, pp. 129–136.

[49] A. Testolin *et al.*, "A machine learning approach to QoE-based video admission control and resource allocation in wireless systems," in *Proc. 13th Annu. Mediterr. Ad Hoc Netw. Workshop (MED HOC NET)*, Piran, Slovenia, Jun. 2014, pp. 31–38.

[50] J. van der Hooft *et al.*, "HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2177–2180, Nov. 2016.

[51] H. X. Nguyen, P. Thiran, and C. Barakat, "On the correlation of TCP traffic in backbone networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Vancouver, BC, Canada, May 2004, pp. V-481–V-484.

[52] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using OpenFlow-assisted adaptive video streaming," in *Proc. ACM/SIGCOMM Workshop Future Human Centric Multimedia Netw. (FhMN)*, Hong Kong, Aug. 2013, pp. 15–20.

[53] A. Mansy, M. Fayed, and M. Ammar, "Network-layer fairness for adaptive video streams," in *Proc. IFIP Netw. Conf.*, Toulouse, France, May 2015, pp. 1–9.

[54] J. W. Kleinrouweler, S. Cabrero, R. van der Mei, and P. Cesar, "Modeling stability and bitrate of network-assisted HTTP adaptive streaming players," in *Proc. 27th Int. Teletraffic Congr. (ITC)*, Ghent, Belgium, Sep. 2015, pp. 177–184.

Matteo Gadaleta received the M.S. degree in electronic engineering from the Polytechnic University of Bari, Italy, in 2014. He is currently pursuing the Ph.D. degree with the Department of Information Engineering (DEI), University of Padova, Italy. His B.S. and M.S. projects entailed the detection of ventricular late potentials in ECG signals and the implementation of an ultralow power wireless sensor network for the monitoring of floods. In 2015, he has been a Research Associate with DEI. His research interests included the design of biometric signal processing techniques for smart wearable devices, and his grant was supported by Samsung Korea through the GRO AWARD Program. His current research topics are centered on the study of human sensing applications. Fields of interests are: signal processing for biomedical signals, deep learning, and applied machine learning.

Federico Chiariotti (S'15) received the bachelor's and master's degrees in telecommunication engineering from the University of Padova, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree. In 2017, he was a Research Intern with Nokia Bell Labs, Dublin. He has authored over 10 papers on wireless networks and the use of artificial intelligence techniques to improve their performance. He was a recipient of the Best Student Paper Award at the International Astronautical Congress in 2015.

Michele Rossi (S'02–M'04–SM'13) is an Associate Professor with the Department of Information Engineering, University of Padova, Italy. In the last few years, he has been actively involved in EU projects on IoT technology (IOT-A, FP7-ICT-2009-5, project no. 257521) and has collaborated with SMEs such as Worldsensing, Barcelona, Spain, in the design of optimized IoT solutions for smart cities and, with large companies such as Samsung and Intel. His current research interests are centered around wireless sensor networks, Internet of Things (IoT), green 5G mobile networks, and wearable computing. Since 2016, he has been collaborating with Intel on the design of IoT protocols exploiting cognition and machine learning, as a part of the INTEL Strategic Research Alliance Research and Development program. His research is also supported by the European Commission through the H2020 ITN SCAVENGE project (MSCA-ITN-ETN, project no. 675891) on green 5G mobile networks. He has authored over 100 scientific papers published in international conferences, book chapters, and journals. He was a recipient of the SAMSUNG GRO Award with a project entitled *Boosting Efficiency in Biometric Signal Processing for Smart Wearable Devices* in 2014 and four best paper awards from the IEEE. He currently serves on the Editorial Board of the IEEE TRANSACTIONS ON MOBILE COMPUTING.

Andrea Zanella (S'98–M'01–SM'13) received the master's degree in computer engineering and the Ph.D. degree in electronic and telecommunications engineering from the University of Padova, Italy, in 1998 and 2000, respectively, where he is an Associate Professor. He has (co)-authored over 130 papers, five books chapters, and three international patents in multiple subjects related to wireless networking and Internet of Things, vehicular networks, cognitive networks, and microfluidic networking. He serves as the Technical Area Editor for the IEEE INTERNET OF THINGS JOURNAL, and as an Associate Editor for the IEEE COMMUNICATIONS SURVEYS & TUTORIALS, and the IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING.