# Just-In-Time Proactive Caching For DASH Video Streaming

Rita Coutinho, Federico Chiariotti, Daniel Zucchetto, Andrea Zanella



# Just-In-Time Proactive Caching For DASH Video Streaming

Rita Coutinho*, Federico Chiariotti†, Daniel Zucchetto†, Andrea Zanella†
*Department of Electrical and Computer Engineering,
Instituto Superior Técnico de Lisboa – Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
† Department of Information Engineering, University of Padova – Via Gradenigo, 6/b, 35131 Padova, Italy
Email: ritavccoutinho@gmail.com, {chiariot, zucchett, zanella}@dei.unipd.it

***Abstract*—Since video traffic has become the major source of generated mobile traffic worldwide, there has been a strong interest in new techniques to improve the Quality of Experience (QoE) of users generating this traffic. Several efforts have been made to propose adaptive streaming algorithms on the client side to be able to adjust the quality of the downloaded video segments to the current network conditions, taking advantage of the Dynamic Adaptive Streaming over HTTP (DASH) standard. This study uses this knowledge to propose a pre-fetching proxy cache, to be placed at the network's edge, which will predict the quality that the client will request for the following segment. The proxy predicts the future network conditions and models the system as a Markov Decision Process (MDP), in order to find the optimal decision for the proxy, given the current network conditions. This Just-in-Time caching technique pre-fetches the segment just before the client requests it, aiming to decrease the total time spent by the client downloading the segments, and indirectly increasing the user's QoE, as the DASH client will perceive better network conditions. The study concludes that the predictive technique improves QoE by increasing the video quality and decreasing the number of stalling events, in comparison to solutions that pre-fetch the previously requested quality, and solutions which do not use any pre-fetching technique.**

## I. Introduction

Mobile video traffic has been growing exponentially in the last years and is currently generating most of the mobile traffic worldwide. By the end of 2021, mobile data traffic is predicted to rise to 49 exabytes, with the share of video traffic rising from 60% (2016) to 78% (2021) [1].

Due to this rapid growth in video traffic, the study of efficient techniques to enable a faster video transmission in order to provide a better Quality of Experience (QoE) to the users is an active field of research with growing interest from the industry.

Several new technologies have appeared for improvement of user QoE, by taking advantage of the MPEG Dynamic Adaptive Streaming over HTTP (DASH) standard [2]; this standard enables breaking a video file down into smaller segments, making several representations of each segment with different qualities and bitrates available in the DASH server. This allows the DASH client to select which adaptation to request from the server for each segment, basing its decision on the network conditions. If the player implements an intelligent adaptation logic, it can prevent the video buffer from emptying out and consequently stopping the playback until more segments have been downloaded.

ISBN 978-3-903176-05-8 

Other examples of adaptive bit rate streaming techniques include Apple's HTTP Live Streaming (HLS) [3], Microsoft Smooth Streaming (MSS) [4] and Adobe's HTTP Dynamic Streaming (HDS) [5]; however, DASH is the only one which was not developed by a specific vendor, and is an international standard. The DASH standard was introduced in 2011 and was later adopted by two major sources of video streaming, Netflix and Youtube.

Caching and pre-fetching are also techniques that have been explored to improve QoE. Caching can be defined as the temporary storage of an object for future use, and pre-fetching can be defined as the action of requesting an object that is expected to be needed in the near future. The introduction of proxy caches in the network has been studied, along with various caching strategies. The purpose of this is to make the network less vulnerable to congestion by making the same content available in places other than the server, so there are fewer requests sent directly to the server, and users can get better performance by streaming the content from the closest available cache. There have also been some attempts to study the impact of pre-fetching on user QoE [6], [7]; however, the cache often needs a considerably large size in order to yield an acceptable cache hit percentage.

The study of optimal bit rate adaptation algorithms used by clients is widely saturated, along with the study of caching and pre-fetching solutions. However, a caching strategy which involves a pre-fetching technique that predicts the client's future behavior is yet to be explored. Therefore, the main objective of this work is to study the impact on the user QoE of the introduction of a predictive proxy cache in the edge of a network, in a connection between a DASH client and a DASH server.

By placing a cache closer to the requesting client, the Round Trip Time (RTT) of the connection will decrease, reducing the time taken by the client to download a video segment. This will induce the client to perceive a better channel capacity, enabling it to request segments with higher qualities. The aim of the proxy cache is to increase the user's QoE by: i) increasing the video quality; ii) reducing the amplitude of the variation in the video quality; iii) reducing the frequency of the video quality variation; iv) reducing the frequency of stalling events; v) reducing the duration of stalling events.

Since real caches have a limited storage capacity, there must be an efficient way to determine which segments to store in the cache over time. To achieve this, a probabilistic approach is considered to predict which video segment the client will

request, given the current network conditions. This prevents the unnecessary storage of unused video segments as well as the unnecessary use of bandwidth to pre-fetch these segments from the server.

The rest of the paper is organized as follows. Sec. II describes the current state of the art in development of pre-fetching and caching techniques, while Sec. III details the system model used in this work, and Sec. IV presents the simulation parameters and the results. Finally, Sec. V concludes the paper with some remarks and possible future extensions.

## II. State of the Art

This section describes the most relevant works published in the literature concerning pre-fetching strategies.

A network awareness study is conducted by Bronzino *et al.* [8], where the authors aim to optimally use the available end-to-end bandwidth by using intermediate nodes to cache video content closer to the client, thus distributing the traffic load over time. Ultimately, the solution aims to improve the QoE for the end user. The proposed solution involves moving the decision on the segment's quality into the network, by introducing a controller and a cache in the edge of the network. In this study, the client will only request the required video segment, and will receive that segment with the bitrate pre-fetched by the controller, which will be stored in the cache. The study takes advantage of the fact that the available bandwidth is easier to predict having a general view of the network infrastructure resources available in the network. Using the information on the client playback and buffer status, the controller exploits the available resources and chooses an appropriate bitrate for the segment it will download to store in the cache. The bitrate selection algorithm used in this study chooses a combination of bitrates for the given sequence of segments to be downloaded at the time (bitrate path). The chosen bitrates are the ones which lead to the highest QoE, given the current network conditions. This algorithm runs within a given time frame, returning the bitrate path with the highest QoE when the time frame ends.

A limitation that is addressed by this study is that in the long run, it is possible that when a new time frame begins, the network conditions will be different from the ones that were considered in the previous time frame, and this may lead to a significant QoE drop if the network conditions are less favorable at this time. Another limitation that can be seen in this study is the fact that the algorithm must be run once for every decision that must be made. Even though the proposed solution introduces some additional costs for the content provider, the authors claim that the resource costs are minimal and that the achieved gain in QoE outweighs the computational costs.

In [6], Liang *et al.* combine both caching and pre-fetching to improve the performance in terms of byte-hit ratio and video bit rates. The architecture of this solution is composed of three modules: a cache manager, a pre-fetch manager and a request pool. The cache manager handles all user requests and video segments received from the content server. If there is a cache-miss, it sends a request to the request pool, which in turn forwards the request to the content server. The cache manager also generates pre-fetch requests for every user request, and sends those to the pre-fetch manager. It only generates pre-fetch requests for successive video segments with the same bit rate as the current request. When the video segment arrives, if the cache is full, the cache manager makes a decision on whether to keep or discard the segment, based on its utility. The pre-fetch manager decides whether the received pre-fetch request should be sent to the request pool or not, based on its current usage.

Evaluation is done by comparing to three alternatives: Least Recently Used (LRU)-based caching approach, a popularity-based caching approach called Popular Content (PC) which caches the top 100 most popular in advance, and an aggressive pre-fetching approach. The study shows that the Average Per-User Throughput (APUT) is 50% higher compared to LRU and PC, and 31% higher when compared to the aggressive pre-fetching, for a cache size of 1GB. In terms of byte-hit ratio, the architecture improves the performance by nearly 84% compared to the aggressive pre-fetching, and a performance gain between 5 and 8 times larger when compared to LRU and PC. This study, however, does not take into account the volatile behaviour of the channel, as it always chooses to pre-fetch the same bit rate as was requested previously.

In [7], Krishnappa et al. investigate the advantages of having a pre-fetching and caching scheme for Hulu (a free hosting service of professionally created video for films and TV shows). The pre-fetching scheme is based on caching the most popular videos of the week provided by the Hulu website. It is compared to the conventional LRU caching. Results show that this yields a hit ratio of up to 77.69% but requires a storage of 236 GB. When evaluating the performance of pre-fetching the popular videos list, it is noted that a maximum hit ratio of 44.2% is obtained when pre-fetching 100 videos, corresponding to a cache storage of 10GB. For the same storage space, the LRU caching scheme yields a hit ratio of 45.53%; however, in this case 5767 videos are downloaded, compared to only 100 when pre-fetching.

Binging is a new trend which has also been studied. Binging is when a user watches multiple episodes of a television programme in rapid succession, typically by means of DVDs or digital streaming. For example, Claeys *et al.* [9] take advantage of the recent trend. Studies show that users stream on average 2.3 episodes per viewing, and so 57% of the streaming sessions could be announced in advance by a proxy. If these announcements were to be made, it would enable a simple prediction for future segment requests and subsequent episodes could be cached in advance, allowing for an improved QoE.

The evaluation of this study is based on the byte hit ratio. It notes an increase in performance of 54% in comparison to the LRU caching strategy. A limitation on this approach is that it does not take into account the possibility of the user ending the session before the episode ends. If this were to happen, many segments would be stored in cache with no purpose, as they would not be served to the client. Also, the bandwidth that would be used to pre-fetch these segments could have been used to serve other clients.

Zhang *et al.* [10] present a dependency-aware caching algorithm which takes into account a dynamic network condition. The study assumes multiple users and multiple requests per


Figure 1. Schematic of the considered scenario.

user, and therefore aims to improve QoE for all users generally and not for a specific user. The algorithm is based on the profit of caching a certain segment, which is defined by the increase in utility of caching that segment. The utility of caching a segment depends on the available bandwidth, the segment size, the number of active client sessions and the number of requests per session. The algorithm decides to cache segments in descending order of profit, and depending on how full the cache storage is.

Although pre-fetching is beginning to attract interest from the research community, to the best of our knowledge there are no works studying optimal prediction-based pre-fetching schemes.

## III. System Model

As depicted in Fig. 1, we consider a DASH client streaming a video by downloading consecutive segments from a server. A proxy is placed between the client and the server, intercepting the client's segment requests and answering them directly if the chosen segment is present in its cache. We assume that the proxy proactively tries to predict the next segment that the client will request and pre-fetches it from the server; if the proxy pre-fetches and stores the correct adaptation, the latency the client experiences is far lower, improving the user QoE.

We denote by $a_c(n)$ the adaptation chosen by the client for the $n$-th segment, and by $a_p(n)$ the one pre-fetched by the proxy. We can distinguish two different scenarios:

- *Cache hit*: if $a_p(n-1) = a_c(n)$, and the proxy has finished downloading the pre-fetched segment when the client's request arrives, the client downloads the segment from the proxy, which requests the next predicted adaptation to the server at the same time;
- *Cache miss*: if $a_p(n-1) \neq a_c(n)$ or the pre-fetching is not complete, the client's request is forwarded to the server, and either the proxy abstains from pre-fetching the next segment (resulting in another cache miss) or its pre-fetching will be in direct competition for the link between server and proxy with the client's download. In any case, the client is prioritized with respect to the proxy, so the latter can only use the bandwidth that is not used by the client.

We can model the scenario above as a Markov Decision Process (MDP), a class of Markovian model defined by a state space $S$ and an action space $A$, both finite and discrete, a state transition matrix $M$ whose elements are the transition probabilities between states $s_n$ and $s_{n+1}$, and a reward function $r(s_n, s_{n+1}, a_p(n))$.

The action space of the problem is represented by the proxy's pre-fetching choices $a_p(n)$: the solution to the problem is the policy $\Pi^* : S \longrightarrow A$ which maximizes the expected reward function for the next step. Note that, as explained above, refraining from pre-fetching a segment is a valid action and should be included in the problem definition. The objective of the proxy is to maximize the client's reward function, which depends on the user QoE for the video; we also assume that the proxy knows the adaptation logic the client is running and can then predict the client's actions in any given situation, in order to preserve the Markov property. In order to model the problem as an MDP, we need to define the reward function, the system state and the transition matrix.

### A. 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 simply chose the bitrate as a proxy for picture quality, but the framework supports any objective QoE metric, which could be pre-computed by the server and served to the client along with the Media Presentation Description (MPD) or computed live using an appropriately trained deep neural network [11].

We finally define the reward function for the pre-fetched segment $n$ as follows:

$$r(q_{n-1}, q_n, \phi_n) = q_n - \beta \|q_n - q_{n-1}\| - \gamma \phi_n, \qquad (1)$$

where $\phi_n$ is an indicator variable that is equal to 1 in case a rebuffering event happens. The first term on the right-hand side accounts for the benefit of a higher quality $q_n$ of the video, while the following two negative terms are penalty factors due to *quality variations* in consecutive frames and *rebuffering events*, respectively. The coefficients $\beta$ and $\gamma$ are weighting factors that regulate the relative importance of the three penalty terms. Note that the structure of the reward function (1) was first proposed and validated by De Vriendt *et al.* in [12], and is used as a comprehensive QoE metric by several algorithms in the literature [13]–[15].

The weights $\beta$ and $\gamma$ 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.

We define the optimal policy $\Pi^* : S \longrightarrow A$ as the policy that maximizes the expected value $E[r_n | s_n, \Pi^*]$ in any state.

### B. MDP definition

Since $q_n$ is directly involved in the reward calculation, its value should be included in the state definition in order to fit the definition of the MDP. Two other parameters indirectly affect the state transitions and the reward: the buffer level and the capacity $C_n$ experienced by the client. We denote the buffer level at the beginning of the $n$-th segment download as $B_n$,

and include it in the state definition. The other parameter, the capacity, is composed of three values: the capacity of the link between client and proxy $C_{CP}(n)$, the capacity of the link between proxy and server $C_{PS}(n)$, and an indicator variable $H_n$ which is equal to 1 in case of a cache hit and 0 otherwise. If $H_n = 1$, $C_n = C_{CP}(n)$, while in the cache miss scenario $C_n = \min(C_{CP}(n), C_{PS}(n))$.

The complete state of the MDP is then a 5-tuple: $s_n = (B_n, q_n, C_{CP}(n), C_{PS}(n), H_n)$. We assume here that the policy implemented by the client is known to the proxy, which should be able to determine the probability distribution of the client's actions in any given state $s_n$.

### C. Small-scale model

The definition of an MDP to represent the video streaming scenario has one major issue: since the download of different segments in different network conditions will take different amounts of time, the Markovian assumption can not be justified without some extra steps.

In order to overcome this problem, we model the capacity of the links between client and proxy ($C_{CP}$) and between proxy and server ($C_{PS}$) as two independent Markov processes, with a timestep $T$ which should be far smaller than the average download time of a segment. If we denote the number of undelivered bits in the segment at step $t$ as $b_t$, we get:

$$b_{t+1} = b_t - C_t T, \tag{2}$$

where $C_t$ is the capacity experienced by the client. We define as $T_{\text{setup}}$ the number of timesteps that the client needs to send the request and receive the response from the server or proxy, during which no useful bits can be downloaded; since the capacity of the client is Markovian, we can calculate the probability that an adaptation $a_c(n)$ will take $N$ timesteps, given the initial capacities and the proxy's action:

$$P(N(a_c(n)) = N | C_t^{CP}, C_t^{PS}, T_{\text{setup}}, H_n, a_p(n)) = \sum_{\mathbf{C}} p(\mathbf{C}) \delta\left(F(a_c(n)), C\right), \tag{3}$$

where $F(a_c(n))$ is the frame size for adaptation $a_c(n)$, $\mathbf{C}$ is the vector of channel capacities experienced by the client from time $t + T_{\text{setup}} + 1$ to $t + N$, and $C$ is the sum of all its elements. The solution to this equation can be computed recursively for any initial combination $(N, F(a_c(n)), C_t^{CP}, C_t^{PS})$ and stored for later use. The time from one client decision to the next is simply $TN(a_c(n))$.

In order to get a cache hit for segment $n + 1$, two conditions have to be met: the proxy has to correctly predict the adaptation the client will require, and it has to download the segment before the client requests it (i.e., the proxy download needs to take $M < N$ timesteps). The probability of the latter can be computed as:

$$P(M(a_p(n)) = M | C_t^{PS}, H_n) = \sum_{\mathbf{C}} p(\mathbf{C}) \delta\left(F(a_p(n)), C\right), \tag{4}$$

where $\mathbf{C}$ now indicates the vector of proxy-server channel capacities from time $t + T_{\text{setup}} + 1$ to time $t + M$. In this case, the capacity for the proxy is equivalent to $C_{PS}(t)$ if $H_n = 1$, and $C_{PS}(n) - \min(C_{CP}(n), C_{PS}(n))$ otherwise, as it has to share the link with the client. In the latter scenario, $M$ and $N$ are not independent, and so must be computed together.

In this way, the large-scale transition from one state $s_n = (B_n, q_{n-1}, C_{CP}(n), C_{PS}(n), H_n)$ to the next $s_{n+1} = (B_{n+1}, q_n, C_{CP}(n+1), C_{PS}(n+1), H_{n+1})$ can be modeled as a Markov process; the joint computation of all variables yields the transition matrix of the MDP.

### D. Solution

Since the problem has a finite horizon, it is possible to solve the MDP analytically:

$$a_p^*(s_n) = \underset{a_p \in A}{\operatorname{argmax}} \sum_{s_{n+1} \in S} E\left[r_{n+1} | s_{n+1}\right] P(s_{n+1} | s_n, a_p). \tag{5}$$

The two parts of the equation can be simply determined:

$$E[R_n | s_n] = \sum_{N=0}^{\infty} r(q_{n-1}, q(a_c(n)), u(NT - B_n)) P(N(a_c(n) = N | s_n), \tag{6}$$

where $u(\cdot)$ is the stepwise function. The last term can be calculated using (3), which was derived from the small-scale channel model. The probability $P(s_{n+1} | s_n, a_p(n))$ is computed directly from the small-scale model in Sec. III-C. The expected reward from all the actions in all states can be then computed, and a simple $argmax$ operation is enough to determine the optimal policy.

This brute-force policy calculation is computationally feasible for problems of relatively limited size and with a short-term temporal horizon; for larger and more long-term oriented problems, solutions such as reinforcement learning can be considered, but this is beyond the scope of this work and will be analyzed in a future extension of it.

## IV. Results

This section presents a simulative comparison between the previously described proactive pre-fetching proxy, a pre-fetching proxy that pre-fetches the next segment having the same quality as the one previously downloaded by the client, and the scenario with no proxy. We use a client running the algorithm described in [15].

### A. Simulation scenario

Our simulation scenario is simple: a proxy is placed between a DASH client and a server, with two independent channels between the client and the proxy and between the proxy and the server. The first has an RTT of 50 ms, while the second has an RTT of 200 ms. As described in Sec. III-C, the two channels are modeled as independent Markov processes with transition matrices $\pi_{C,PS}$ and $\pi_{C,CP}$, defined as

$$\pi_C = \begin{bmatrix} 0.9 & 0.1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0.05 & 0.9 & 0.05 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0.05 & 0.9 & 0.05 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0.9 & 0.05 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0.05 & 0.9 & 0.05 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0.1 & 0.9 \end{bmatrix}.$$

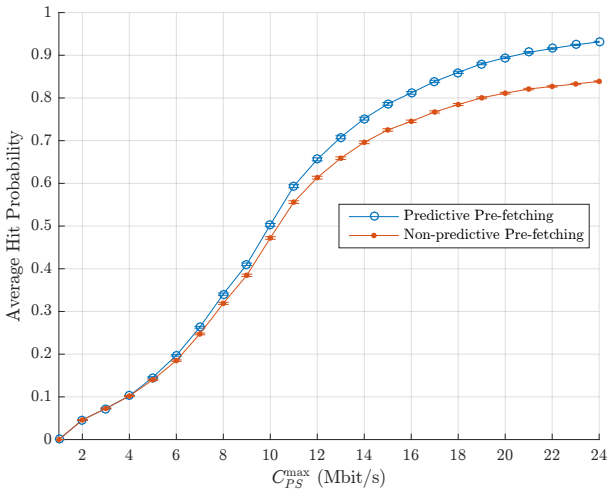


Figure 2. Average hit probability, with 95% confidence intervals.

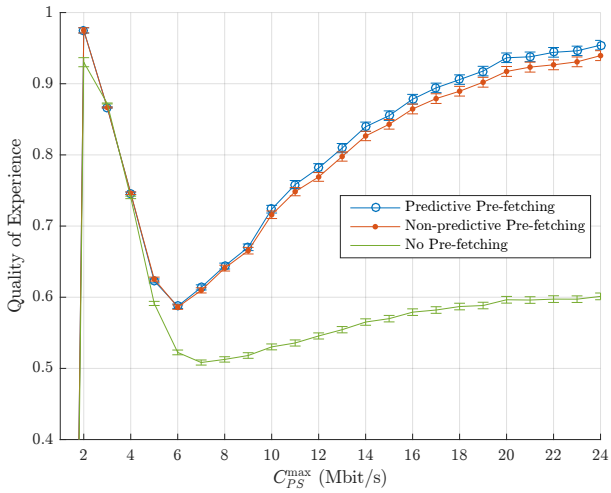


Figure 3. QoE, with 95% confidence intervals.

The size of this square matrix is different for the client-proxy and proxy-server channels. In particular, we define the states to represent the link capacity measured in Mbit/s, starting from 1 up to a maximum channel capacity. The maximum channel capacity for the client-proxy link is set to $C_{CP}^{\max} = 6$ Mbit/s, so the six states for the client-proxy channel model correspond to available capacities of $\{1, 2, 3, 4, 5, 6\}$ Mbit/s. The same holds for the proxy-server link; however, its maximum capacity $C_{PS}^{\max}$ has been varied from 1 Mbit/s to 24 Mbit/s to explore different scenarios, including the ones where the proxy-server link capacity is actually lower than the client-proxy capacity, which is a challenging setting for a prefetching proxy. The time step $T$ for the small scale model was set to 100 ms.

In the following simulations we use $\beta = 6$ and $\gamma = 10$ as the weighting factors in the reward function in (1).

### B. Hit probability

First, we analyze the average cache hit probability when varying the maximum proxy-server link capacity (Fig. 2).

For very low values of $C_{PS}^{\max}$ it is unlikely that the proxy is able to pre-fetch segments before the client requests them. In order to better explain this statement, consider the scenario where the next segment requested by the client has the same quality as the currently downloaded one. For the proxy to be able to successfully pre-fetch the next segment, the proxy-server bandwidth available to the proxy has to be at least equal to the client-proxy bandwidth. As soon as a cache miss happens, since the proxy-server capacity is low, the client requests the segment from the server using almost all of the available bandwidth, thus preventing the proxy to download the correct quality for the next segment. This causes an avalanche of cache misses, which lower the average hit probability.

Instead, when the maximum proxy-server link capacity is high, the hit probability is large, particularly for the predictive proxy. This proves that the predictive proxy is able to accurately predict the next segment quality even when the number of states in the channel model, and, therefore, its capacity variability, is large. As expected, the same quality pre-fetching strategy offers an inferior performance. Since the proxy pre-fetches the different qualities in order of how likely they are to be requested, as the maximum proxy-server capacity approaches to infinity, the proxy is able to pre-fetch any segment quality without considering the capacity limitation, thus yielding an average hit probability close to 1.

### C. Average QoE

In order to measure the overall QoE, we use a metric proposed in [15]. This metric is a linear combination of the average video quality $\bar{q}$ and its standard deviation $\sigma_q$, both normalized by the maximum available quality $q_{\max}$, and a parameter $F$ that models the influence of stalling events.

$$QoE = 5.67\frac{\bar{q}}{q_{\max}} - 6.72\frac{\sigma_q}{q_{\max}} - 4.95 \cdot F + 0.17\,, \qquad (7)$$

with $F$ defined as

$$F = \frac{7}{8}\max\left(\frac{\log(\phi)}{6} + 1, 0\right) + \frac{1}{8} \cdot \frac{\min(\psi, 15)}{15}\,, \qquad (8)$$

where $\phi$ is the frequency of stalling events and $\psi$ is their average duration. Fig. 3 shows a large increase in QoE by using a caching proxy and, in particular, by using a predictive pre-fetching strategy.

For low proxy-server maximum capacity, the client, as explained earlier, is almost always downloading the segment from the server. With such a low proxy-server capacity, the channel model has very few states, causing a channel with low capacity but also low variability. In this scenario, the client adaptation algorithm is able to predict the channel very well, thus avoiding quality switches and rebuffering, offering large QoE even when playing low quality segments.

From there, increasing the proxy-server maximum capacity brings higher variability to the channel, while still forcing the download of low quality segments. This decreases the QoE value up to a point, which in our scenario is $C_{PS}^{\max} = 6$ Mbit/s, where the large channel variability is compenstaed by the possibility to play high quality segments. From there, the QoE increases with the maximum proxy-server capacity.

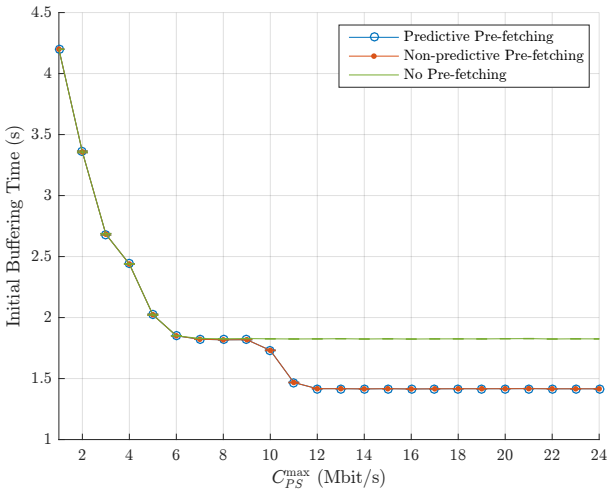


Figure 4. Initial buffering time, with 95% confidence intervals.

### D. Initial buffering time

The time needed to fill the buffer before the actual start of the playout is not considered in the QoE metric plotted in Fig. 3. Therefore, for completeness, we show its behavior in Fig. 4 for different maximum proxy-server capacities. We consider the playout to start, and therefore the initial buffering period to end, when the buffer contains the first 3 seconds of video. We observe that, for $C_{PS}^{\max}$ lower than 9 Mbit/s, the performance with and without the proxy is the same, because of the low hit probability which forces the client to download the segment directly from the server. Instead, for larger maximum proxy-server capacity values the client benefits from the presence of a proxy, reducing the initial buffering time. The initial delay stabilizes for $C_{PS}^{\max} \approx 2C_{CP}^{\max}$, since, in this case, the probability of the proxy-server link being the bottleneck is almost zero.

### E. Advantages of the scheme

In conclusion, the benefit of using a pre-fetching proxy is clear: Fig. 3 shows a significant increase in the QoE for both pre-fetching schemes. The predictive pre-fetching scheme increases the QoE only slightly with respect to the simple non-predictive scheme, but its advantage lies in the significantly higher hit rate: since every cache miss means that a segment is downloaded by the proxy but never used by the client, increasing the hit rate greatly improves the efficiency of the system. A predictive pre-fetching system, installed before the "last mile" in a DSL or cellular network, can help improve end users' QoE without imposing a significant additional load on the core network.

## V. Conclusions

In this work, we modeled a DASH video streaming scenario with the help of a proactive pre-fetching proxy as an MDP, and found the optimal strategy for the proxy.

Although the benefits in terms of QoE are not huge when compared to a simpler proxy which pre-fetches the next segment at the same quality as the one currently streamed by the client, the combined increase in QoE and cache hit rate means that the predictive proxy can exploit its better awareness of the scenario to provide a better quality while wasting less bandwidth and reducing the load on the server.

Future developments of this work include a thorough testing with different client algorithms and a more realistic scenario, as well as the development of a proxy that can maximize the long-term QoE instead of just the instantaneous one by using reinforcement learning.

## References


[1] T. J. Barnett, A. Sumits, S. Jain, and U. Andra, "Cisco Visual Networking Index (VNI) Update Global Mobile Data Traffic Forecast," *Vni*, pp. 2015–2020, 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] I. Sodagar, "ISO/IEC FDIS 23009-4: Dynamic Adaptive Streaming over HTTP (DASH)," International Organization for Standardization, Geneva, Switzerland, Standard, 2013.

[3] M. Pantos, "IETF Draft: Apple HTTP Live Streaming (HLS)," Internet Engineering Task Force, Standard, 2013.

[4] A. Zambelli, "Iis smooth streaming technical overview," *Microsoft Corporation*, vol. 3, p. 40, 2009.

[5] Adobe HTTP Dynamic Streaming (HDS). [Online]. Available: http://www.adobe.com/devnet/hds.html

[6] K. Liang, J. Hao, R. Zimmermann, and D. K. Y. Yau, "Integrated prefetching and caching for adaptive video streaming over HTTP," *Proceedings of the 6th ACM Multimedia Systems Conference on - MMSys '15*, pp. 142–152, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2713168.2713181

[7] D. K. Krishnappa, S. Khemmarat, L. Gao, and M. Zink, "On the feasibility of prefetching and caching for online TV services: A measurement study on hulu," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6579 LNCS, pp. 72–80, 2011.

[8] F. Bronzino, D. Stojadinovic, C. Westphal, and D. Raychaudhuri, "Exploiting network awareness to enhance dash over wireless," in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2016, pp. 1092–1100.

[9] M. Claeys, N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, S. Latre, and F. De Turck, "An Announcement-based Caching Approach for Video-on-Demand Streaming," *Network and Service Management (CNSM), 2015 11th International Conference on*, 2015.

[10] C. Zhang, J. Liu, F. Chen, Y. Cui, and E. C. H. Ngai, "Dependency-aware caching for HTTP Adaptive Streaming," *2016 Digital Media Industry and Academic Forum, DMIAF 2016 - Proceedings*, pp. 89–93, 2016.

[11] M. Zorzi, A. Zanella, A. Testolin, M. D. F. D. 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.

[12] J. De Vriendt, D. De Vleeschauwer, and D. Robinson, "Model for estimating QoE of video delivered using HTTP adaptive streaming," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, Ghent, Belgium, May 2013.

[13] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-DASH: a deep Q-learning framework for DASH video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. PP, no. 99, Oct. 2017. [Online]. Available: https://doi.org/10.1109/TCCN.2017.2758370

[14] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 325–338, 2015.

[15] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck, "QoE-driven rate adaptation heuristic for fair adaptive video streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 2, pp. 28:1–28:24, Mar. 2016.