Exploiting Network Awareness to Enhance DASH Over Wireless

Francesco Bronzino[§], Dragoslav Stojadinovic[§], Cedric Westphal[†], Dipankar Raychaudhuri[§]

[§]WINLAB, ECE Department, Rutgers University, New Brunswick, NJ, USA
[†]Innovation Center, Huawei Technology, Santa Clara, CA, USA
[†]Computer Engineering Department, University of California, Santa Cruz, CA, USA
Email: [§]{bronzino,stojadin,ray}@winlab.rutgers.edu, [†]cwestphal@huawei.com

***Abstract*—The introduction of Dynamic Adaptive Streaming over HTTP (DASH) helped reduce the consumption of resources in video delivery, but its client-based rate adaptation is unable to optimally use the available end-to-end network bandwidth. We consider the problem of optimizing the delivery of video content to mobile clients while meeting the constraints imposed by the available network resources. Observing the bandwidth available in the network's two main components, core network, transferring the video from the servers to edge nodes close to the client, and the edge network, which is in charge of transferring the content to the user via wireless links, we aim to find an optimal solution by exploiting the predictability of future user requests of sequential video segments, as well as the knowledge of available infrastructural resources at the core and edge wireless networks in a given future time window. Instead of regarding the bottleneck of the end-to-end connection as our throughput, we distribute the traffic load over time and use intermediate nodes between the server and the client for buffering video content to achieve higher throughput, and ultimately significantly improve the Quality of Experience for the end user in comparison with current solutions.**

I. Introduction

Video consumption has grown so fast that the core network is now congested and has become the bottleneck during peak hours. Recently, [1] shows that video rates from a large CDN decrease during prime time owing to the congestion in the network. The share of video streaming is expected to increase to 79% by 2018 [2], further exacerbating the issue. Internet Service Providers are struggling to provide high quality services to their customers due to their inability to allocate enough capacity to meet such demand, especially at peak hours. While in the past, core network was over-provisioned when compared with the wireless edge, the trend is for both wired and wireless to align owing to the growth in bandwidth demand, the increased capacity of wireless technologies, and the economical incentive for operators to not build over-capacity inside their core networks under tremendous bandwidth pressure.

To adjust the demand to the network conditions, Dynamic Adaptive Streaming over HTTP (DASH) has been introduced. DASH lets the client adjust the rate of the video stream by monitoring the network conditions perceived by this stream. DASH is a client-based rate adaptation, as the server is stateless and the network is considered as a black box. The client measures the end-to-end throughput between itself and the server and modifies the rate accordingly.

While this approach has been incredibly successful and DASH-like rate adaptation now accounts for most of the video traffic (say, Netflix or YouTube), end-to-end rate adaptation is suboptimal. Indeed, end-to-end monitoring measures the lowest throughput of all the links in between the client and the server, while the available bandwidth on these links varies a lot.

As a simple illustrative example, consider the following scenario of Figure 1: the client is connected to the server by two links (say, a wireless link for the edge network, and a wired link to the server). Both of the links' bandwidth will oscillate and in modern networks, *both are congested* [1]. Therefore an end-to-end mechanism will yield the minimal available bandwidth of each link. If for one unit of time, the capacity of the wireless is 1 unit of transfer, while the capacity of the wired link is 2, and for the next unit of time the capacities are reversed, then the end-to-end mechanism can only achieve a rate of 1.

However, inserting an intermediary node with storage capacity in between the two links could increase the throughput significantly. In the illustrative example, during the first time slot, the full capacity of the wired link could be used to deliver 1 unit to the client and 1 unit to the intermediate storage; in the second time slot, this extra stored unit can be delivered over the air interface, for a total delivery of 2 units. On average, the delivery is of 1.5 for each time slot, versus 1 in the end-to-end mechanism. While this example is simplistic, and does not take into account rate adaptation, it shows the benefit of a network-assisted content delivery mechanism. Here we present an intermediated rate adaptation mechanism which leverages the different time-varying capacities of the multiple links in the network.

The goal of our work is to propose a novel infrastructural approach to control the load on the network for clients with quickly changing available capacity while maintaining high quality experience for the end users. The experience is defined as a set of quality metrics, such as average bitrate, temporal variability and amount of rebuffering time. Our solution is based on two key ideas: (1) to use local caches strategically deployed into the access networks to decouple the transmission

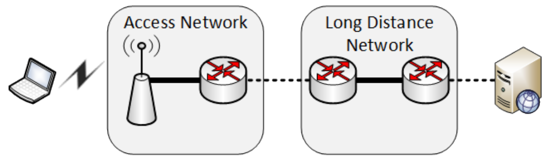


Fig. 1: Basic network representation.

speed of the Internet long distance network from the local access network; and (2) to exploit the predictability of future available infrastructural resources (at the core and edge wireless network) and the predictability of video requests to distribute the load on the network of the video streams and control the Quality of Experience for the end users.

The rest of the paper is structured as follows: in Section II we will discuss our contributions in comparison to other related works; in Section III we will present the network model that we use to then design our solution; Section IV will provide a detailed experimental analysis of our design through extensive simulations and finally Section V will conclude the paper.

II. Related work

Bianchi [3] has considered the role of intermediate storage in content delivery, however in the context of ATM and without rate adaptation. They did observe a similar benefit in their application scenario. Since that work, Dynamic Adaptive Streaming over HTTP (DASH) [4] has been introduced and successfully deployed as a standard for video streaming.

Proxy-assisted caching and prefetching has been widely studied in the literature. Some approaches consider the quality of the connections [5] and the usage of the client buffers [6]. Approaches for transcoding proxy caching are also presented in [7], [8] in which the proxy caches different versions of the content to handle the heterogeneous user requirements. Prefix caching [9] caches only the frames at the beginning to minimize the average initial delay. Exponential segmentation [10] divides the video object such that the beginning of a video is cached as a smaller segment. The lazy segmentation approach [11] determines the segment length according to the user access record at the late time. [11] discusses a segment-based proxy pre-fetching for streaming delivery. [12] evaluates the 1-ahead, n-ahead, and priority-based segment prefetching. The results show that if the bottleneck link is between client and proxy, all prefetching schemes achieve high cache hit rate after 2-3 clients requesting a video. On the other hand, if the bottleneck link is between proxy and server, no prefetching helps. Our approach considers the link between the cache and the server and makes a pre-fetching decision accordingly.

Proxy technologies have also been used to enhance QoE. [13] uses information about wireless channel quality at the base station to provide QoE and fairness among clients. [14] proposes WiDASH, a proxy for adaptive HTTP streaming over wireless networks that implements a quadratic linear optimization problem to decide on the rate to use for each user/segment while giving higher priority to lower video rates.

Alternatives to intermediate storage and proxies have also been considered in order to improve DASH, or video streaming in general. [15] proposes a Network-Friendly DASH (NF-DASH) architecture for coordinating peer-assisted CDNs operated by ISPs and the traditional CDNs. [16] considers the interaction of DASH and caching. [17] analyzes the waiting time and network utilization with service prioritization considering both on-demand fetching/caching and prefetching in a P2P-assisted video streaming system. [18] formulates the CDN assignment as an optimization problem targeting minimizing the cost based on the QoE constraints to CDN servers at different locations. [19] shows by measurement the shortcomings of today's video delivery infrastructure and proposes a control plane in the network for video distribution for global resource optimization. Hybrid P2P-CDN video streaming enhancements have also been considered [20], [21] (serving content from dedicated CDN servers using P2P technology) as have improvements through telco-CDN federation (CDNs operated by telecommunication companies, enabling users to reach CDN caches that are closer). Telco-CDN federation can reduce the provisioning cost by 95%. Using P2P can lead up to 87.5% bandwidth savings for the CDN during peak access hours.

Prediction of content requests or available resources has been looked from different perspectives in the literature. [22] has looked at how to predict the request from the users by looking at social interactions. Here, we use the natural predictability that video streaming offers. [23] describes a web prefetching module running on the CDN main node (controller) which downloads web content that will be requested in the future to the LAN CDN surrogates. Our work is different as the time domain is much smaller and the prefetching utilizes the bandwidth more dynamically. In addition, [23] does not specifically consider video content. [24] tries to optimize the utilization of the wireless channel to delivery buffered video (i.e. streaming with availability of buffer at the receiving side) jointly with the minimization of the rebuffering time using a bandwidth prediction model, but does not consider network caching or adaptive video streaming.

In conclusion, to the best of our knowledge, our paper is the first work to exploit the predictability of future available infrastructural resources (both at the core and edge wireless network) and the predictability of video requests to distribute the load on the network of the video streams and control the Quality of Experience for end users.

III. Network Model and Architecture

A. Overview

We build our system around the network model shown in Figure 2 from [3] (and similar to [25], [26], [27]): in this classical scenario, a client connects to the Internet through a local access network; the goal of the client is to retrieve video content from a remote server that can be reached by traversing a long distance network. We focus on a scenario where mobile devices are connected to the access network through a wireless interface (or more than one); our system

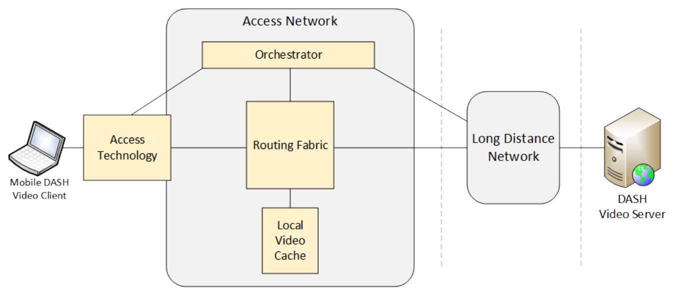


Fig. 2: System model.

is designed to be technology agnostic so both wifi or cellular connectivity can be considered.

We base our video delivery model around Dynamic Adaptive Streaming over HTTP (DASH). In this protocol each video file is divided into segments of equal duration encoded at different bitrates and all stored in one or more web servers. The choice of using DASH as the underlying technology for our system naturally derives from its popularity as it relies on the already available HTTP infrastructure of web servers, proxies and caches. In a typical DASH system, a client interested in retrieving the video first has to retrieve a Media Presentation Description file that contains information on the structure of the video, the available different bitrates for the segments and at which location they are stored, from the server. Once this step is completed, the client proceeds to sequentially download consecutive video segments. When a new segment has to be retrieved, the client selects an appropriate bitrate where the decision is usually based on different factors such as recently experienced bandwidth [28] or current buffer size [29].

This model heavily relies on the ability of the client to estimate the available bandwidth and general network resources, a task arguably very difficult under normal network conditions [30], and even more complex under wireless and mobile environments due to the high dynamicity caused by time-varying fading, shadowing interference and hand-off delays [31], [32]. For this reason, we propose to move the adaptation logic where this information is more easily accessible: into the network. In our system, a caching system is positioned at the edge of the architecture[1]. An orchestration system, that from now on we will call the network controller and that could be centralized or distributed implements the adaptation logic by means of periodically scheduling video segment requests to temporarily store the corresponding video segments into the cache; to do so, we assume that the controller, at the beginning of a periodic time window of given size, schedules the download of video segments to the available caches where the clients will then retrieve them from. The scheduling process is based on two fundamental pieces of information to which the controller has access:

A general view of the network infrastructure resources availability both in the core network and the edge network: as different studies have proved, it is indeed possible to predict the variability of users connectivity in wireless networks to a certain extent by either exploiting movement predictability [33] or by recognizing performance patterns [34], which is even easier if done from a convenient location inside the access network[2].

Information on the client playback and buffer status: assuming that the normal DASH client behavior is expected (i.e. sequential downloading of consecutive video segments), the controller maintains a detailed view of the current status of active video streams by tracking the progression of the requested segments; this includes the size of the video buffers available at each client and the playback time.

We now analyze the details of each protocol and algorithm implemented in our system; we start from describing how the basic DASH model would be modified in order to exploit the new functionalities in Section III-B; we then provide details on the optimization functions involved in the scheduling process in Section III-C and finally, Section III-D introduces the scheduling algorithm performed by the controller.

B. System Protocols

Our solution's goal is to maximize the Quality of Experience (QoE) for the end users while only exploiting the available resources. We use the concept of a controller that orchestrates the necessary operations; while we present it as a centralized solution, we claim that the same results could be obtained with a distributed solution, such as a web proxy, as our algorithm would work in either setting. Moreover, while we refer to this orchestration system as the controller, we do not limit our solution to Software Defined Networks: any technology able to track HTTP requests can be applied; possible solutions include the use of an HTTP proxy that manipulates the traffic or obviously the use of a centralized SDN controller. Our system relies only on tracking and exploiting available in-network resources (i.e. capacity and caches) and tracking and eventually modifying HTTP requests from the clients. We then abstract the required actions into three main steps.

1) Stream initialization. In order to initialize the streaming process, each client has to request the DASH MPD file from the server. The controller captures these requests and retrieves the same information either by deep packet inspection of the returned content or by requesting the same file from the webserver. With this information, the controller obtains a complete view of the video that can potentially be retrieved. To simplify the description and our notation, we assume that each video is composed of N segments $\boldsymbol{s_i^N} = \{s_1, s_2, ..., s_N\}$, each of duration S seconds and available at M different bitrates.

2) Bitrate Adaptation. Once the MPD file is retrieved, the client can proceed to sequentially request the video segments; while in a normal DASH setup, the client would be in charge of selecting one of the available bitrates for each requested segment, we leave this duty to the edge controller; this design

[1]The cache should be within the wireless operator's network, but not necessarily co-located with the base station, to allow for easy cell level mobility.

[2]As a matter of fact, our algorithm enhances predictability in the network, as it attempts to keep the rate constant, as we will see in the QoE discussion and in the evaluation results.

choice is justifiable by considering the fact that the controller has the best possible view of the available resources, by accessing infrastructure components and tracking the client process through its issued requests. The controller selects among the different bitrates based on two main factors: status of the video reproduction at the client (i.e. buffer size and previous quality selection) and future availability of infrastructure resources; in particular we assume that the controller has access to information regarding residual capacity available in the network and bandwidth for any specific client for a time window of size T seconds.

3) Streaming process. Following the normal DASH model, clients sequentially retrieve video segments by mean of issuing HTTP requests. We differentiate from the original model by having the clients retrieve the video from the caching system without the need for specifying any particular version of the segment. This could be done using commonly exploited techniques such as: 1) having the network controller modify the originally retrieved MPD by replacing content URLs with locally resolvable ones or 2) by transparently capturing the HTTP requests (e.g. if the controller is implemented as an HTTP proxy) or 3) in order to meet the proposed goals the controller divides the delivery path into two components: from the server to the local caches available at the edge network and from the caches to the client over the wireless link. The controller uses the available information in the time slot of size T seconds to provide in time delivery to the caches.

Additional details regarding the involved scheduling algorithm will be provided in section III-D, but in order to do so, we first need to explain the Quality of Experience employed.

C. Optimization Function

1) QoE Model: We do not re-invent the wheel for QoE and use a typical QoE model that takes into account average rate, rate variations and buffering events. We present the model here only to describe how it is computed in the algorithm.

Average quality: multiple works have shown how the average video quality can not be used as the sole metric in determining the quality experienced by a user. Nevertheless, we still need to have a way of factoring the proportional quality between different available bitrates; for this reason we refer to previous work [35] where it has been widely proven that there is a direct relationship between bitrate selection and quality governed by logarithmic laws.

Temporal quality variance: different studies have proven how representation switching can factor negatively against the quality of experience; in particular, [35] found that only up to 0.5 quality switches per minute are considered tolerable by users, causing exponential increase of rate of abandonments if these criteria are not met. Moreover, as suggested in the ITU standard [36], human memory effects can distort quality ratings if noticeable impairments occur in approximately the last 10-15s of the sequence, exponentially decaying afterwards, causing past factors to relatively influence the current visualized video.

Buffering time and ratio: it has been widely demonstrated that frequency and length of video rebuffering highly affects the perceived quality of a video stream where each event increases the rate of abandonment and reduces the probability of client return [35].

While the third factor negatively impacts the abandonment rates from users, we will work with the initial assumption that these events are occurring very infrequently; namely we assume the available bandwidth always provides enough resources to deliver at least the lowest quality of video; a rebuffering cost could then be introduced into our optimization function to factor it into the scheduling algorithm.

We initially focus our attention on the first two points in defining a model of the quality of experience perceived by a client over a video session. First we express the quality of a video segment following the logarithmic law:

$$q(r_i, r) = \alpha \ln \frac{\beta r_i}{r}$$

where r is the minimum quality available for the segment, r_i is the quality of the considered segment and α and β are specific factors that vary depending on the type of the displayed content.

Taking the second factor into consideration, we define a temporal quality variance penalty for two following video segments selected at qualities $q_{i-1}(r_i, r)$ and $q_i(r_j, r)$ as:

$$v_i = \begin{cases} \eta(q_i - q_{i-1}), & \text{if } q_i \geq q_{i-1} \\ -\gamma(q_i - q_{i-1}) & \text{if } q_i < q_{i-1} \end{cases}$$

where η and γ are positive factors that determine how much changes impact the overall experience when a transition to a higher or to a lower quality representation happens.

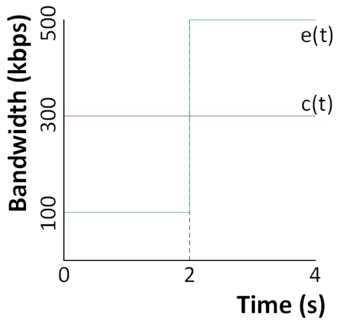
Therefore we can calculate the QoE as the objective function capturing the defined values, where for a sequence of N segments selected at qualities $\boldsymbol{q}_1^N$:

$$\phi(\boldsymbol{q}_1^N) = \sum_{k=1}^{N} (q_k - v_k)$$

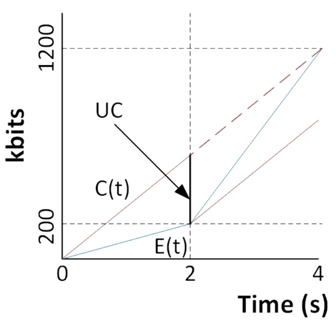
where $v_k = v(\boldsymbol{q}_{k-M}^k)$.

2) Cost Factors Considerations: We can include other metrics in our model to account for the network incentives; for instance, a network operator may have a peering agreement such that it incurs a cost linear with the amount of bandwidth. Therefore we can introduce a cost $c_{bw} = \rho r_i$ with ρ as a price per unit of data transfer; also, there may be a cost associated with the storage of data at the edge cache. Namely, if $\omega_s(t)$ is the amount of data in transit from the servers to the clients which is stored in the edge cache at time t, we can introduce a storage cost $c_s = \kappa \sum_t \omega_s(t)$.

We can then subtract these costs from the utility to get a joint objective which takes into account the QoE of the end-user (or the benefit to the end-user as a potential revenue to the operator) minus the costs associated with the system.



(a)



(b)

Fig. 3: Example on how future knowledge can be used to exploit residual capacity. **(a)** Provides the general scenario and **(b)** its representation through cumulative downloaded data.

Symbol	Meaning
$c(t),\ e(t)$	Capacity over time for core and edge network
$C(t),\ E(t)$	Cumulative function of $c(t)$ (resp. $e(t)$)
$UC(t)$	$C(t)-E(t)$
W	Size of predicted window
$t_{\mathrm{d},i}$	Download deadline of segment i before rebuffering
$\alpha,\ \beta$	QoE factors dependent of displayed content
r	Minimum quality available for video segments
r_i	Bitrate quality with index i
$q(r_i, r)$	Logarithmic law of video quality
$\eta,\ \gamma$	Scaling factors affecting cost of quality transitions
v_i	Temporal quality variance penalty
$\boldsymbol{q}_1^N$	Sequence of N selected video segments qualities
$\phi(\boldsymbol{q}_1^N)$	QoE objective function

TABLE I: Summary of symbols

D. Rate and Streaming Algorithms

As discussed at the beginning of this section, we consider four main blocks as shown in Figure 2: a web server, that has complete availability of all the segments composing the video; a long distance network, an access network that uses a caching system in order to support the streaming process; finally, a mobile client attached to the network through a wireless interface[3]. Resources available on the two logical hops between the server and the client are regulated by the capacity function $c(t)$ in the core, and the capacity function $e(t)$ at the edge. $c(.)$ and $e(.)$ are time varying instantaneous capacities. They are assumed to be varying with a coherence time such that knowledge of $c(.)$ (resp. $e(.)$) until time t allows the controller to estimate $c(s)$ (resp. $e(s)$) for the time window $s \in (t, t+W)$, for some time W.

Let us first introduce a supporting example to build intuition behind the presented model. In contrast to current solutions where the logic employed to perform the bitrate adaptation process is based on estimation of available resources perceived in the recent past, we consider a partially anticipative case in which a finite window of future edge and core network capacity variations are known beforehand. This information is used by the network controller to schedule which video segments to download in the upcoming future, by either transferring the content directly the server to the client, or by using the available caches as support.

Consider the simple scenario represented in Figure 3a, where $W = 4$s and $c(t)$ is constant at 300 kbps and $e(t)$ varies from the first 2 seconds period at 100 kbps to the second period at 500 kbps. In this context the client wants to retrieve a video divided in segments of size 2 seconds and available at three different representations: 100 kbps, 300 kbps and 600 kbps. Initially the video buffer is empty, so let us assume that the first segment of video will be downloaded at minimum quality to reduce the startup wait time for the client. This corresponds to downloading 200 kbits of data which in our example, given the initial bottleneck of the wireless link will take 2 seconds to complete. Once this happens, the video client has 2 seconds of video available for display and it should try to download the next segment by this amount of time otherwise a rebuffering event would occur (i.e. the video client remains stuck waiting for more video data to be available).

[3]While more than one access interface could be considered, we first focus our analysis on a single access interface.

We denote by t_{d} the download deadline, that is the amount of time until the buffer runs out and the next segment needs to be fully downloaded for display. t_{d} in our example will then occur at the time of 4 seconds. In current scenarios, where downloads are controlled by the client, only two representations would meet the deadline: 100 kbps (which would allow for more segments to be downloaded) and 300 kbps. In our anticipative case, we know that even though the first download will be bandwidth limited by the current bottleneck, we are actually under-utilizing the network resources as 200 kbps of unused capacity have not been exploited during the first segment download. As we are scheduling downloads in advance, when evaluating the amount of time needed to download next segment we consider the unused capacity as data that might actually have been downloaded to the local caches. The actual amount of data that could have been downloaded to a location in the network (i.e. the caches) is easily calculated using the cumulative versions of $e(t)$ and $c(t)$, that we will call $E(t)$ and $C(t)$, as:

$$UC(t) = C(t) - E(t)$$

In our particular case, $UC(t)$ is highlighted in black in Figure 3b and corresponds to 400 kbits. Now we can assume that this amount of data could have already been transferred close to the edge network preventing the core network from being the bottleneck in the second period. This difference can again be noticed in Figure 3b: given that the amount of downloaded data always corresponds to the minimum of the two functions $E(t)$ and $C(t)$ (as a client can never download more than what is allowed by the bottleneck in the network), at the time of 2 seconds this would correspond to 200 kbits. Assuming no preemptive caching is applied, the amount of data that could be downloaded in the second period is again delimited by the minimum of the two lines and in this case it would correspond to the plain line representing $C(t)$. If we assume that the core network has moved an amount of data corresponding to the unused capacity to the caches, the minimum has now to be taken between $E(t)$ and the dotted version of $C(t)$.

Our algorithm applies these concepts of deadlines and evaluations of unused capacity to explore the state space of valid combinations of segment bitrate downloads to select the one in the time window that produces the highest QoE utility value described in Section III-C with rebuffering events never occurring (hence always meeting the set deadlines).

Bitrate Selection Algorithm. We now formalize the provided concepts; refer to Table I for a summary of all used notation; we define the streaming process as a combination of two tasks for each window of future knowledge: first the controller **schedules segments downloads** given the available predicted resources and the client reproduction status; second the **delivery of the segments** to the the mobile hosts with use of the edge caches. The core of the scheduling algorithm is based on a recursive function performed at the beginning of each time window as described in Algorithm 1; this function searches for an optimal path (sequence of segment bitrates) among all possible combinations. Starting from time t and given a potential bitrate j for the next required segment with index i, the controller determines the effect of such representation on the download process and, once the entire future knowledge window is consumed, determines the QoE of the selected path. The function calculates the download time for the given segment using the available throughput and residual capacity from previous steps. Each time the function is called from the base algorithm, the starting residual capacity is assumed to be zero. In recursive calls, the residual capacity in the core network, if available according to the known cumulative throughput functions C and E, may be adjusted. As long as the knowledge window limit is not reached, the recursion follows. Once the end of the window (or potentially the end of the video) is reached, the utility function value of the current path is calculated and compared with the best path previously found and only the better of the two is kept. In case a rebuffering time event is detected, the path is declared invalid and the function returns. Our algorithm can be summarized in four main steps:
1) Avoid overrunning the client buffer by waiting until some space is created (lines 9-17).
2) Calculate the download deadline for the considered segment ($t_{\mathrm{d},i}$) and, given the previously accumulated unused capacity, evaluate if the deadline can be met by calculating the amount of time necessary to transfer the required data ($t_{\mathrm{e},i}$); if not, return the previously found best path (lines 18-24).
3) If the future window has been consumed or the end of the file is reached return the $GREATEST$ between the current path and the previous best path based on their utility value (lines 25-27 and 35-37).
4) Otherwise recursively evaluate the same function for the next segment (index $i+1$) over its possible representations. (lines 28 to 34).

After completing the recursive process, the sequence of segments with the highest QoE value is returned and the controller can use it to instruct the other components on how to proceed (i.e. instruct the clients and the caches on how to proceed for downloading the selected segments).

Algorithm 1 Path building algorithm

```
1:  function FINDOPTIMALPATH($i, j, UC, W_\mathrm{r}, B, D_\mathrm{t}, \phi_\mathrm{best}$)
2:      // $i$ - next segment index
3:      // $j$ - assumed bitrate of the next segment
4:      // $UC$ - available residual capacity
5:      // $W_\mathrm{r}$ - remaining time in knowledge window
6:      // $B$ video buffer available at client
7:      // $D_\mathrm{t}$ time to complete displayed segment
8:      // $\phi_{best}$ - current best path found
9:      if $B$ == max_buffer_size then
10:         wait until the end of current displayed segment
11:         if $D_\mathrm{t} > W_\mathrm{r}$ then
12:             return GREATEST($\phi_\mathrm{best}$, current_path)
13:         else
14:             reduce $B$ by segment size
15:             $D_\mathrm{t} \leftarrow$ segment size
16:         end if
17:     end if
18:     $UC \leftarrow$ increment given waited time
19:     $t_{\mathrm{e},i} \leftarrow$ segment download time
20:     if $t_{\mathrm{e},i} < W_\mathrm{r}$ then
21:         if $t_{\mathrm{e},i} > B + D_\mathrm{t}$ then
22:             // Rebuffering time > 0
23:             return $\phi_\mathrm{best}$
24:         end if
25:         if $i == N$ then
26:             // Reached end of video
27:             return GREATEST($\phi_\mathrm{best}$, current_path)
28:         else
29:             $W_\mathrm{r} \leftarrow W_\mathrm{r} - t_{\mathrm{e},i}$
30:             update $B$, $D_\mathrm{t}$ and $UC$ given $t_{\mathrm{e},i}$
31:             for all bitrates $m$ do
32:                 FINDOPTIMAL-PATH($i+1, m, UC, W_\mathrm{r}, B, D_\mathrm{t}, \phi_\mathrm{best}$)
33:             end for
34:         end if
35:     else
36:         return GREATEST($\phi_\mathrm{best}$, current_path)
37:     end if
38: end function
```

Interleaving windows. The main characteristic of our algorithm is that it greedily tries to use as many resources as are available in the time window without much consideration for the following time slots. Since the QoE cost of switching to a higher bitrate is lower than the cost of switching to lower bitrates, the consequence of such behavior is the tendency to select higher bitrates toward the end of the window to consume the available remaining capacity. This is not always the optimal path in the longer run, since it might be necessary to choose a lower bitrate at the beginning of the next time window, and thus suffer the QoE drop due to switching to a lower bitrate immediately afterwards. To avoid this problem, we consider an alternative solution: while we still apply the same algorithm for the complete window of size W to select the best path, we only apply the obtained optimal path until an earlier moment $W - t_i$, where t_i is smaller than W. This way we prevent a higher bitrate from being selected in the last t_i of the time window W, avoiding possible quality drops at the beginning of the next window. After this is done, the new considered window will start from time $W - t_i$.

Optimality. Our algorithm is optimal as it exhaustively considers all possible rate trajectories under the available estimated rates and returns the one which maximizes the objective function. This is possible owing to the fact that the set of rates is relatively limited, and that we can predict the future rates with accuracy only for a relatively short window of time, therefore the number of potential rate combinations to compute is not prohibitive. The algorithm can be made less computationally intensive by taking into account some considerations: (1) as quality transitions (in particular negative ones) negatively affect the final utility value, we can set a limit on the number of these events; in particular, the number of rate transitions cannot be more than the utility of the highest rate minus the utility of the lowest rate divided by the penalty of a rate transition; (2) if the bottleneck bandwidth is always higher than a certain bitrate during the duration of a time window, all video representations with lower bitrate can be left out of consideration.

IV. SIMULATIONS

In this section, we evaluate the gains achieved by our system and joint rate adaptation through a set of Matlab based simulations. In order to understand its potential, we implemented the core logic of our system and compared it to the behavior of common DASH implementations. While different proprietary algorithms are used in some of the available commercial solutions (e.g. Apple HLS, Adobe HDS, etc.), we implement our baseline following the behavior in the logic implemented by Netflix-like video services, which can be summarized in two main characteristics:

- the DASH client downloads and keeps only video segments for the following t seconds of playback at any given time (i.e. the buffer size is limited by time, not data space);
- the DASH client logic adapts video quality by a moving average of the data rate estimates experienced on the previous k segments delivered (we set k to 5 for our experiments).

While it is easy to identify a wide selection of factors that might affect the final results of our simulations, we try to fairly compare the baseline with the two variants of our algorithm by applying the same conditions for each run (i.e. evolution of network infrastructure resources during the experiment time). In the next two subsections we will describe the model used to characterize these resources and the video data set employed in our tests.

Video Dataset and QoE Model. For this set of simulations, we used a video content of 5 minutes of length. The video is divided into 2 second long segments, with each segment available in three different bitrate representations (1 Mbps, 400 kbps and 100 kbps for the first case, 2 Mbps, 1.2 Mbps and 300 kbps for the second). While our system supports Variable Bitrate for the video segments, we use only constant bitrates for these simulations (i.e. all segments at the same quality level have the same size). The Quality of Experience is calculated following the description provided in Section III-C, using parameters: $\alpha = 1$, $\beta = 1$, $\gamma = 1 = \eta = 0.1$.

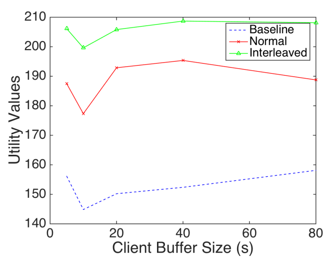
System Resources and Network Model. In our simulations we do not take availability of video segments at different servers into consideration; we use a single server that has the desired video content available at all times. Moreover, we do not consider any limit in the cache size of the intermediate nodes. We provide results using two different network models, one synthetic, one trace-based:
- first, we model the available network bandwidth as a classical two finite-state, discrete-time Markov chains, where transitions occur at constant times, every 2 seconds, and transitions occur only between the two nearest states; this is done to try to capture slow variations attributable to client mobility (for the wireless links) and evolution in congestion for the core network. The Markov chains used to produce the presented results are shown here, but we obtain similar results for a wide range of transition matrices and rate parameters:

$$P_{\mathrm{E}} = \begin{bmatrix} 0.5 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.33 & 0.33 & 0.33 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.33 & 0.33 & 0.33 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.33 & 0.33 & 0.33 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.4 & 0.4 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.7 & 0.3 \end{bmatrix} R_{\mathrm{E}} = \begin{bmatrix} 100 \\ 300 \\ 500 \\ 700 \\ 900 \\ 2300 \end{bmatrix}$$

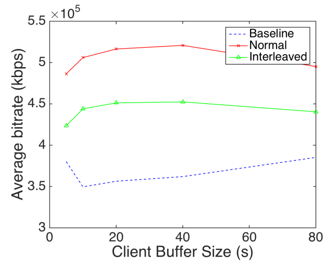
$$P_{\mathrm{C}} = \begin{bmatrix} 0.5 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.33 & 0.33 & 0.33 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.33 & 0.33 & 0.33 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.33 & 0.33 & 0.33 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.33 & 0.33 & 0.33 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.5 \end{bmatrix} R_{\mathrm{C}} = \begin{bmatrix} 700 \\ 900 \\ 1100 \\ 1100 \\ 1100 \\ 1300 \end{bmatrix}$$

where P_{E} represents the transition matrix for the wireless edge access network where for each state the corresponding bandwidth is shown in R_{E} (expressed in kbps); the same values are shown for the core network in P_{C} and R_{C}.
- second, we use a real cellular trace, collected and presented in [37]. This trace consists of multiple days worth of data collected in a real metropolitan environment in a European city, using a HSDPA modem while commuting with public transportation. The core network is modeled similarly to the previous case, using the same transition matrix but with the following values for the available throughput: $R_{\mathrm{C}} = [1000; 1200; 1400; 1600; 1800; 2000]^T$.

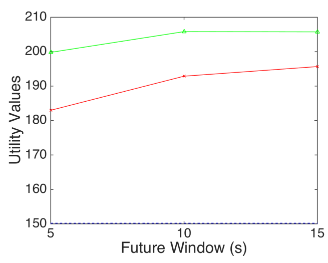
Results. We evaluate our system under two varying factors: video buffer size available to the DASH client and window of



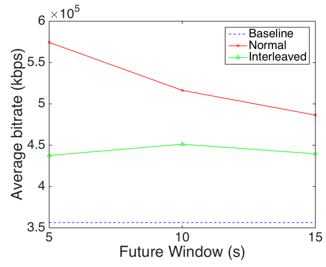
(a) Utility value with varying buffer size



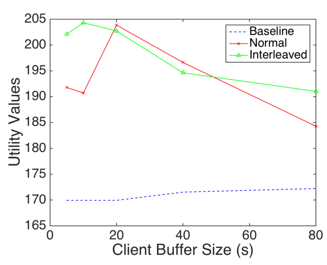
(b) Average bitrate with varying buffer size



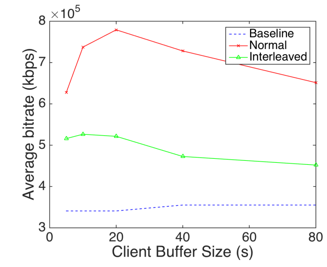
(c) Utility value with varying future window size



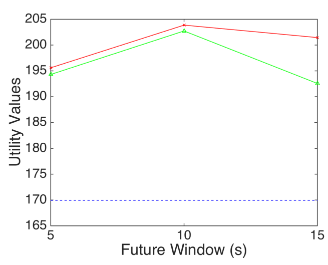
(d) Average bitrate with varying future window size



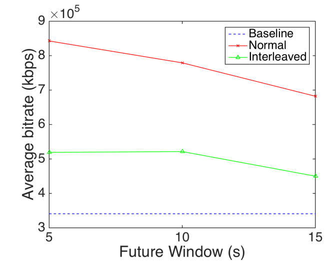
(e) Utility value with varying buffer size



(f) Average bitrate with varying buffer size



(g) Utility value with varying future window size



(h) Average bitrate with varying future window size

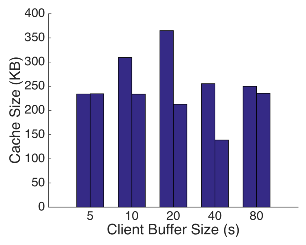
Fig. 4: Simulation results. Figures a-d provide results for a Markov chain based wireless model, while Figures e-h use real world traces collected in an urban environment.

future knowledge of available bandwidth at the two analyzed network components. The first factor varies between 5, 10, 20, 40 and 80 seconds. The second between 5, 10 and 15 seconds; we selected 5 seconds as the base for this metric as smaller values would not provide enough future information to take full advantage of the proposed algorithm (although we still notice quality improvements over such scenarios) and 15 seconds as the biggest value as bigger values would cause the potential space of the solution to explode making the proposed algorithm too computational intensive. Figure 4 collects all the obtained results. For each of the data points represented, we repeated the experiment 5 times and collected the average result. This does not apply for the baseline in Figures 4g and 4h as the future window size can vary only for our algorithms; for these cases we used a single data point using client buffer size of 20 seconds. In general, the results confirm the overall benefit of our system with gains of at least 15% points in the utility value for both our algorithm versions. This not only corresponds to a more stable experience (as variations, in particular decreases in quality, strongly affect the final QoE value), but also in a higher average bitrate quality for all the experiments analyzed.

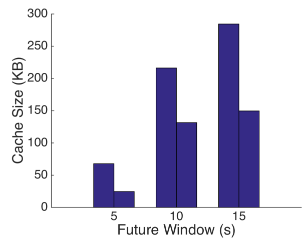
As expected, the buffer size available for the clients is not a major influencing factor for the analyzed use case, as neither algorithm uses the buffer size information to modify its adaptation logic; we expect this parameter to gain more importance for longer videos, where a low bandwidth period might be better compensated by accumulated buffer.

More interestingly, we can notice that increasing future knowledge window size also increases the computational cost of scheduled delivery time deadlines, since there are now longer periods of time that are considered in the calculation. The achieved gains in this case might not justify the additional processing time costs.

Comparing our two proposed algorithms we notice an important result: the interleaved solution outperforms the normal solution in terms of utility but gets outperformed in terms of average bitrate; this is due to the greedy behavior of the normal algorithm explained in the design section. These results confirm our initial guess: that the interleaved solution is better at maintaining an average video bitrate closer to the highest available video bitrate less than the average bitrate experienced in the access network, providing a more stable viewing experience for the user of the system.



(a)



(b)

Fig. 5: Max buffer occupancy over different runs.

Figure 5 shows the employed memory usage at the network caches. In particular, it shows the cache size for the single client experiment of the first analyzed case. This value enables us to understand the resource cost of running our system, and is of particular interest to service providers that would implement such a caching scheme as scalability is directly affected by the consumed resources. The amount of video data buffered at the cache nodes using our algorithm ranged from

0 up to 365 kB, meaning that it would scale well for larger videos and higher number of clients and not impose a high resource cost for the provider of the service.

V. Conclusions

As network congestion can arise from either the core or the wireless edge (as observed in current networks under the stress of video streaming applications[1]), we have presented a system that takes advantage of a cache in between these two. Our model is based on DASH and assumes that bandwidth consumption and network congestion has a coherence time that allows to predict capacity over a short time window. Compared with DASH, our system improves the usage of the available end-to-end throughput between the server and the client. We optimize the video delivery method, utilizing the intermediate nodes between the server that has the whole video content available, and the client, as an intermediate video content cache. This enables us to increase the effective throughput in certain parts of the network path, and to improve the end-user QoE.

We evaluated the gains achieved by using our system with a set of Matlab simulations. We implemented our algorithm and compared it with the behavior of DASH implementations against synthetic and real throughput traces. Our evaluations demonstrate that our method of video delivery improves the Quality of Experience for the end user by keeping the bitrate more stable and achieving significantly higher average bitrates than the benchmarks. While it does introduce some additional costs for the content provider, we claim that the resource costs are minimal, and the achieved gain outweighs the computational costs.

References

[1] P. Fiadino, A. D'Alconzo, A. Bar, A. Finamore, and P. Casas. On the detection of network traffic anomalies in content delivery network services. In *Teletraffic Congress (ITC), 2014 26th International*, pages 1–9. IEEE, 2014.

[2] Cisco Visual Networking Index: Forecast and Methodology, 20132018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html/.

[3] G. Bianchi and R. Melen. The role of local storage in supporting video retrieval services on atm networks. *IEEE/ACM ToN*, 5(6), 1997.

[4] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *MultiMedia, IEEE*, 18(4):62–67, 2011.

[5] S. Jin, A. Bestavros, and A. Iyengar. Network-aware partial caching for internet streaming media. *Multimedia systems*, 9(4):386–396, 2003.

[6] Z. Miao and A. Ortega. Proxy caching for efficient video services over the internet. In *Packet Video Workshop (PVW'99)*, 1999.

[7] K. Li, K. Tajima, and H. Shen. Cache replacement for transcoding proxy caching. In *IEEE/WIC/ACM Web Intelligence'05*, pages 500–507, 2005.

[8] W. Qu, K. Li, H. Shen, Y. Jin, and T. Nanya. The cache replacement problem for multimedia object caching. In *Semantics, Knowledge and Grid, 2005. SKG'05.*, pages 26–26. IEEE, 2005.

[9] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *IEEE INFOCOM'99*, volume 3, pages 1310–1319, 1999.

[10] S. Park, E. Lim, and K. Chung. Popularity-based partial caching for VOD systems using a proxy server. In *IEEE International Parallel & Distributed Processing Symposium*, page 115, 2001.

[11] S. Chen, H. Wang, X. Zhang, B. Shen, and S. Wee. Segment-based proxy caching for internet streaming media delivery. *MultiMedia, IEEE*, 12(3):59–67, 2005.

[12] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Helping hand or hidden hurdle: Proxy-assisted http-based adaptive streaming performance. In *Proc. IEEE MASCOTS*, 2013.

[13] Z. Yan, J. Xue, and C. W. Chen. QoE continuum driven HTTP adaptive streaming over multi-client wireless networks. In *IEEE ICME'14*.

[14] W. Pu, Z. Zou, and C. W. Chen. Video adaptation proxy for wireless dynamic adaptive streaming over http. In *Packet Video Workshop*. IEEE, 2012.

[15] Z. Li, M. K. Sbai, Y. Hadjadj-Aoul, A. Gravey, D. Alliez, J. Garnier, G. Madec, G. Simon, and K. Singh. Network friendly video distribution. In *Int. Conf. on the Network of the Future*, volume 1, 2012.

[16] R. Grandl, K. Su, and C. Westphal. On the interaction of adaptive video streaming with content-centric networking. *Packet Video Workshop*, 2013.

[17] F. Liu, B. Li, and H. Jin. Peer-assisted on-demand streaming: characterizing demands and optimizing supplies. *IEEE Trans on Computers*, 2013.

[18] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian. Optimizing cost and performance for content multihoming. *ACM SIGCOMM CCR*, 42(4):371–382, 2012.

[19] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A case for a coordinated internet video control plane. In *ACM SIGCOMM'12*, pages 359–370, 2012.

[20] C. Huang, A. Wang, J. Li, and K. W. Ross. Understanding hybrid cdn-p2p: why limelight needs its own red swoosh. In *ACM NOSSDAV'08*, pages 75–80. ACM, 2008.

[21] F. Bronzino, R. Gaeta, M. Grangetto, and G. Pau. An adaptive hybrid cdn/p2p solution for content delivery networks. In *VCIP*, 2012.

[22] F. Malandrino, M. Kurant, A. Markopoulou, C. Westphal, and U.C. Kozat. Proactive seeding for information cascades in cellular networks. In *IEEE INFOCOM'12*, pages 1719–1727. IEEE, 2012.

[23] L.R. Ariyasinghe, C. Wickramasinghe, P.M.A.B. Samarakoon, U.B.P. Perera, R.A. P. Buddhika, and M.N. Wijesundara. Distributed local area content delivery approach with heuristic based web prefetching. In *IEEE ICCSE'13*, pages 377–382, 2013.

[24] Z. Lu and G. de Veciana. Optimizing stored video delivery for mobile networks: the value of knowing the future. In *INFOCOM, 2013 Proceedings IEEE*, pages 2706–2714. IEEE, 2013.

[25] A. Chanda, C. Westphal, and D. Raychaudhuri. Content based traffic engineering in software defined information centric networks. In *IEEE Infocom NOMEN workshop*, 2013.

[26] A. Chanda and C. Westphal. Contentflow: Mapping content to flows in software defined networks. In *IEEE Globecom*, 2013.

[27] Y. Yu, F. Bronzino, R. Fan, C. Westphal, and Mario Gerla. Congestion-aware edge caching for adaptive video streaming in information-centric networks. In *IEEE CCNC*, 2015.

[28] T. Stockhammer. Dynamic adaptive streaming over HTTP: standards and design principles. In *ACM conf on Multimedia systems*, 2011.

[29] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. ACM SIGCOMM*, 2014.

[30] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *ACM IMC*, 2012.

[31] C. Müller, S. Lederer, and C. Timmerer. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *ACM Workshop on Mobile Video*, 2012.

[32] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. A comparison of quality scheduling in commercial adaptive HTTP streaming solutions on a 3g network. In *ACM Workshop on Mobile Video*, 2012.

[33] A. J. Nicholson and B. D. Noble. Breadcrumbs: forecasting mobile connectivity. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 46–57. ACM, 2008.

[34] Q. Xu, S. Mehrotra, Z. Mao, and J. Li. Proteus: network performance forecast for real-time, interactive mobile applications. In *ACM MobiSys*, 2013.

[35] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A quest for an internet video quality-of-experience metric. In *Proceedings of the 11th ACM Workshop HotNets*, pages 97–102. ACM, 2012.

[36] Methodology for the subjective assessment of the quality of television pictures. Recommendation ITU-R BT.500-13, 2012.

[37] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Commute path bandwidth traces from 3G networks: analysis and applications. In *ACM Multimedia Systems Conference*, 2013.