To DASH, or not to DASH? Optimal Video Bitrate Selection and Edge Network Caching in MEC-empowered Slice-Enabled Networks

Dionysis Xenakis

***Abstract*—An increasing body of works highlight that Dynamic adaptive streaming over HTTP (DASH) can negatively impact the user Quality of Experience (QoE) in cache-enabled networks. Besides, network-agnostic DASH cannot fully harness the potential of slice-enabled 5G and Beyond networks, which can promise minimum rate performance for the i) MEC-empowered cellular base station (BS) to user link, and ii) content delivery network (CDN) to MEC-empowered BS link. In this paper, we show that network-agnostic DASH is incompatible with MEC-empowered edge network caching and network slicing in next generation mobile data networks. Accordingly, we propose joint video bitrate selection and edge network caching when specific rate and cache size guarantees can be provided along the end-to-end content delivery path. By formulating the aforementioned problem as a dynamic program (DP), we present an exhaustive (brute-force) search algorithm to optimally solve it and derive an exact algorithm of polynomial time to avoid unnecessary recalculations of previously visited solution branches. Through extensive experimental results we assess the performance of the proposed algorithm and compare it against other solutions, aiming to draw valuable insights for algorithmic design tailored to MEC-empowered slice-enabled mobile data networks.**

***Index Terms*—Dynamic Adaptive Streaming over HTTP, Edge Caching, Network Slicing, 5G, 6G, Dynamic Programming**

I. Introduction

MULTI-access edge computing (MEC) and Network Slicing (NS) are integral parts of 5G and Beyond 5G (B5G) networks, which aim to the delivery of ultra-rich media formats with strict Quality of Service (QoS) and Quality of Experience (QoE) performance guarantees. MEC-empowered services promise to greatly reduce network response times and decongest end-to-end (e2e) backhaul links by leveraging compute and storage resources at the network edge. NS deliver logically partitioned network islands that bind together heterogeneous resources (spectrum, compute, storage, content) across different operational domains, e.g., mobile network operators (MNOs), infrastructure, MEC and over-the-top (OTT) service providers. Fully and jointly harnessing the potential of MEC and NS is critical towards the standardization and research for 5G and B5G mobile data networks [1]–[3].

Video streaming constitutes the largest volume of mobile data traffic (>75% by 2023), being a key focus area of improvement for service providers [4]. Recent reports highlight that network responsiveness upon video streaming is critical for user satisfaction. Besides, the Quality of Experience (QoE) is primarily affected by the time-to-content and the video stalls encountered during the video play-out [5]. Dynamic Adaptive Streaming over HTTP (DASH) is currently the most popular method for delivering video content over mobile networks [6]. According to DASH, the original video content is transcoded into multiple video bitrates and is partitioned into a number of video segments that have an equal playout time. Video segments of different resolutions are fully aligned in time and are stored at the Content Delivery Network (CDN), to enable video consumers switch among the available resolutions on a segment-by-segment basis. DASH has served the mobile video delivery well in pre-5G network setups, where the e2e network status was primarily driven by fluctuations of the wireless channel and the backhaul links. However, in view of 5G/B5G mobile data networks that can bind together heterogeneous communication, compute, storage and network resources [7], [8], network-agnostic DASH inevitably lacks the intelligence necessary to fully harness the potential of MEC and NS.

To bridge the gap between MEC-empowered content caching, e2e network slicing, and application-layer DASH, in this work, we bring to light and address the problem of "oscillation dynamics" in MEC-empowered slice-enabled data networks. In particular, we focus on the emerging scenario where the mobile data network can dedicate storage capacity at the MEC-empowered proxy BSs and reserve e2e network resources between i) the MEC-empowered proxy BS and the user (*BS-to-user link*) and ii) the CDN and the MEC-empowered proxy BS (*CDN-to-BS link*). Through experimentation, we highlight that, if not properly designed, the intermediation of MEC-empowered proxy BSs along the e2e CDN-to-user streaming path can greatly increase the time-to-content upon cache miss events and result in under-utilization of dedicated backhaul resources upon cache hit events, if traditional DASH is deployed. Accordingly, we present a novel video streaming over HTTP paradigm where the MEC-empowered BSs cache a carefully planned sequence of video segments to allow full utilization of the CDN-to-BS backhaul link while delivering available content through the BS-to-user link. We argue that given knowledge on i) the video segments cached at the MEC-empowered BS proxy, and ii) the storage/rate performance guarantees set for the e2e link, we can readily identify the maximum video bitrate that the mobile data network can support; thus, enabling multiple video segment requests for the same video file and user, better utilization of available/reserved resources, and complete mitigation of video

D. Xenakis (Corresponding Author) is Assistant Professor at the Department of Digital Industry Technologies of the National and Kapodistrian University of Athens, Greece. E-mail: nio@uoa.gr.

stalls. Our key contributions can be summarized as follows:

- We provide an experimental study that brings to light key performance limitations and design pitfalls of application-driven video streaming over HTTP in cache- and slice-enabled mobile data networks.
- We formulate the joint bitrate selection, content placement and storage allocation problem in MEC-empowered slice-enabled mobile data networks under fixed buffer and rate performance constraints.
- We derive an exact (optimal) algorithm to solve the original problem in polynomial time and fully analyze its time/memory complexity requirements.
- We provide a comprehensive study to assess the performance of the proposed algorithm and highlight the key trade-offs inherent to mobile video streaming over HTTP in MEC-empowered slice-enabled setups.

At this point, we note that the proposed modeling and algorithm are aligned with recent specifications provided by the European Telecommunications Standards Institute (ETSI) on MEC [2], [3], a.k.a., ETSI MEC, and 3GPP on the 5G system [7]. In more detail, group specification [2] provides implementation details and discusses the different roles that the ETSI MEC and the 3GPP 5G functional components should undertake in joint MEC/5G deployments. According to it, MEC can be flexibly deployed in different locations ranging from near the base station to the central Data Network. Four different deployment options are provisioned, the first of which considers co-deployment of MEC applications into the 5G base station. This particular deployment option is commonly considered in current literature and is also studied in this work. Common for all deployment options is the use of a 5G User Plan Function (UPF) instance to steer traffic between the targeted MEC applications and the network. Besides, group specification [3] describes different use cases and service scenarios including, among others, the effective support of content caching, bandwidth management, and SAND through joint optimization of the ETSI MEC and the 3GPP cellular infrastructures, i.e., Sections A.3, A.16 and A.27, respectively. Our work aims to fully utilize such technological capabilities, providing the necessary decision support framework for joint content caching and video bitrate selection given specific network and storage capacity guarantees/constraints.

To the remainder of the paper, Section II summarizes related works. Section III provides an experimental study on "oscillation dynamics" created by DASH in MEC-/slice-enabled data networks. Section IV provides the proposed problem formulation and solution, along with the required time/memory complexity analysis. Section V includes an in-depth experimental study on the performance of the proposed solution, drawing valuable design guidelines for mobile video streaming in MEC-empowered slice-enabled 5G/B5G networks.

II. Related Works and Motivation

Video traffic is proliferated by massive requests for the same video content, making a very small portion of it to be popular [9]. In previous work in [10], we have addressed the problem of cooperative content caching in 5G network clusters composed by MEC-empowered BSs of different cache sizes. Assuming a common content popularity distribution across each cluster, we have formulated cluster-wide content placement as a zero-one multiple knapsack problem (ZOMKP) where the cluster-wide cache hit probability is to be maximized. An exact (optimal) algorithm that solves the ZOMKP has been provided and extensive simulation results have compared the performance of the proposed algorithm with greedy and random caching. Under the assumption that each BS stores a single video file and DASH delivers entire files (not segments), the authors in [11] assess the performance of random video caching and multi-casting in large-scale stochastic networks. By leveraging stochastic geometry, tractable expressions for the successful transmission probability in dense networks have been derived. Instead of caching entire popular video files and performing video streaming based solely either on cache hits, or on full utilization of backhaul connectivity (cache miss events), in this work we identify the sequence of segments of a given video file that should be cached in order to minimize the required cache size at the MEC-empowered BS given specific rate performance guarantees for the e2e video streaming path.

In [12], the authors discuss the problem of oscillation dynamics created by DASH in Information Centric Networks (ICNs) and propose bitrate-based partitioning of available caches (higher bitrates are stored closer to the users). Assuming that each user is served through a single video streaming path and that all video files are partitioned into the same number of segments, they formulate the cache placement problem as a binary integer linear problem (BILP) and propose distributed heuristics to solve it in polynomial time. In our work, we extend the valuable insights derived in this work by bringing to light the problem of oscillation dynamics created by DASH over MEC-empowered slice-enabled networks, where specific rate performance guarantees can be provided across the e2e video streaming path.

Enhanced predictions of the radio link quality and user mobility have also been shown to improve DASH [13]–[15]. The authors in [13] employ Kalman filters to balance the responsiveness and smoothness of DASH through accurate prediction of the wireless link capacity. Yang et al. [14] propose two machine learning (ML) algorithms that leverage video request and channel state information logs to improve network caching and bitrate selection at the MEC servers. The problem of maximizing the system-wide average video bitrate through QoE-aware RRM is presented in [15]. Although stall and bitrate switch events are not considered, fairness and social welfare are encompassed into a Nash bargaining model that addresses the transmit power allocation problem per user/BS. The performance of server- and network-assisted DASH (SAND) is assessed in [16], assuming co-location of BSs and MEC servers. Aiming to maximize a weighted sum of the average video bitrates, stall ratios and bitrate switches per user, the authors formulate the joint video bitrate selection and MEC load balancing problem as a MINLP. Accordingly, they decompose the original MINLP into two sub-problems that are solved using polynomial time heuristics. The fairness and QoE performance of the proposed algorithms is compared

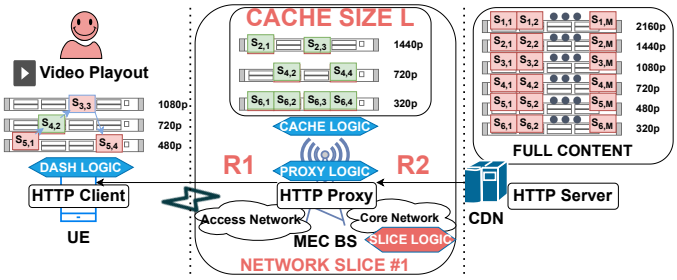


Fig. 1. System Setup and Decision Logic Blocks

against simple client-based DASH heuristics, assuming that MEC servers host the full set of target video segments.

The authors in [17] consider a two-tier cooperative MEC architecture, where a pool of MEC servers maintains a joint cache of non-overlapping video segments and cellular BSs attach to MEC servers to exploit their caching and transcoding capabilities. By integrating input on the RRM strategy of BSs, joint optimization of MEC-empowered caching and transcoding is formulated as a MILP, which is solved using low complexity heuristics. Collaborative video caching and transcoding are investigated in [18], where an ILP is proposed to minimize the initial playback delay of end users. Assuming that MEC servers and cellular BSs are co-located and that each user requests for a fixed video bitrate, the ILP is relaxed into sub-problems solved independently. Different from [13]–[18], this work addresses the problem of joint edge network caching and video bitrate selection in MEC-empowered slice-enabled networks where specific network resources are dedicated to guarantee the CDN-to-BS and BS-to-user rate performance. Our focus is on how to attain the maximum (optimal) cache storage savings while completely avoiding video stalls, without relaxing the original problem, or using meta-heuristics.

To summarize, instead of focusing on on-off scenarios where video streaming is either based on cached content [10]- [11], or on backhaul connectivity [13], we focus on the emerging scenario where specific segments of the full video file can be cached near the edge network to fully leverage dedicated resources along the e2e CDN-to-user streaming path. We also extend prior works on oscillation dynamics [12] in the context of MEC-empowered slice-enabled networks, bringing to light the impact of traditional HTTP proxy strategies. To our knowledge, we address the problem of joint video bitrate selection and edge network caching under specific storage/rate performance guarantees for the first time. Besides, different from the majority of existing works [16], [17], we propose an exact (optimal) algorithm to solve the original joint optimization problem, providing time/memory complexity analysis.

III. The Problem of Oscillation Dynamics in MEC-Empowered Slice-Enabled Networks

Fig. 1 provides an illustrative example of how video streaming over HTTP is performed under a MEC-empowered slice-enabled mobile network. We consider that the user equipment (UE) aims to consume a tagged video file f, all segments of which are available in different resolutions at the CDN. The MEC-empowered cellular BS acts as an intermediary between the user and the CDN server, also reserving e2e resources to seamlessly support the video streaming over HTTP process. The UE is served through network slice #1 that reserves i) up to a storage capacity L at the (serving) MEC-empowered BS for caching specific segments of the target file f, ii) radio resources to guarantee an average throughput of R_1 Mbps for the BS-to-user link and iii) backhaul resources to guarantee an average throughput of R_2 Mbps for the CDN-to-BS link. The slice format parameters L, R_1, and R_2, are selected by the *slice logic* run at the MEC-empowered BS and remain fixed for the entire video session. The MEC-empowered BS further employs i) a *cache logic* to infer on the most appropriate subset of video segments to be cached, and ii) a *proxy logic* that controls how HTTP client requests are served through the BS-to-user link, or the CDN-to-BS and BS-to-user links (cache miss). Video segment requests are adapted based on the DASH logic run at the UE (HTTP Client), specifying the target resolution and sequence ID(s) of requested segments.

A. System setup, video datasets and building blocks

For the purposes of our study, we have implemented all functional and algorithmic components of Fig. 1 at full-scale in Python. Executable files along with a tutorial on our experimental setup are available in [19]. Our setup included three distinct containerized functions, each hosting the functionality of a specific node in Fig. 1 and running on a different Dell OptiPlex 3060 PC (Intel Core i5@3GHz processor, 8GB RAM DDR4 running Ubuntu 22.04 LTS 64-bit). PC1 hosted the VLC player, which has been configured to request video segments from a local HTTP client that places HTTP GET requests according to the DASH logic considered. PC2 hosted the full functionality of the HTTP proxy, which serves cache hits, or places HTTP GET requests for missing video segment to the CDN server (PC3). Communications between PC1-PC2-PC3 where employed over a *physical underlay* network of point-to-point 10Gbps Ethernet links; nonetheless, to realistically emulate the 5G wireless channel in BS-to-user and CDN-to-BS links, we have implemented a *logical overlay* network built on fine-grained application-layer rate control.

To achieve this, prior to the egress (transmission) socket of each HTTP server (i.e., the HTTP proxy at PC2 and the CDN server at PC3), we have implemented a 5G Channel Emulator that counts the bits of egress packets transmitted per sec and matches them with the channel throughput produced by realistic channel emulation traces derived using Matlab. In this fashion, we realistically emulate fluctuations of the instantaneous network throughput in the BS-to-user and CDN-to-user links while simultaneously attaining the target average throughput values of R_1 and R_2 with very high accuracy; thus, alleviating limitations of existing software tools that face difficulties in emulating fine-grained network-level rate-control, e.g., Mininet. This approach further allows for the full control, visibility and reproducibility of the experimental setup, enabling fair comparison between the different algorithms under scope, e.g., by allowing the use of the same seed/traces across different experiments. Besides, the use of

real-life equipment makes it impossible to have exactly the same channel baseline across different experiments.

For benchmarking, we have considered the Big Buck Bunny video with an original resolution at 2160p, frame rate at 30 FPS and video playout duration of $V_f = 634.6$ seconds [20]. By using the H.264 video encoder of FFmpeg [21], we have transcoded the original video file into six resolutions, i.e., 2160p, 1440p, 1080p, 720p, 480p and 360p, and used the GPAC MP4 Box [22] to partition each file into DASH-compatible video/audio segments of a playout time equal to $T = 10$ seconds (last segment is 4.6 secs). Video segments and manifest files are available in [23]. We now specify the experimental study setup in light of Fig. 1.

DASH Logic: We employ the VLC media player [24], which implements DASH in a segment-by-segment fashion, i.e., the next segment is requested only if the previous one is downloaded. We consider the Predictive DASH strategy of VLC, which adapts the video bitrate based on re-buffering events and stall times, to attain higher video bitrates.

Cache Logic: We employ *Random Segment Caching*, i.e., the MEC-empowered cellular BS orders all segments of a target video content randomly (all video bitrates) and caches files sequentially until it fully fills its cache.

Proxy logic: We distinct between two different proxy strategies to highlight the importance of instilling cache-awareness into the HTTP proxy streaming logic:

Send-After-Get (SAG) is the traditional method employed by HTTP proxies (a.k.a., store-and-forward). In cache hits, the SAG proxy delivers the cached video segments through the BS-to-user link directly. In cache misses, the SAG proxy requests the target video segment from the CDN server and downloads/stores the full segment before relaying it to the UE.

Send-While-Get (SWG). In cache misses, it forwards packets received by the CDN server without waiting for the the full video segment to be downloaded/stored (a.k.a. cut-through).

Slice logic: We consider a fixed network slice reserving up to $L = 150$ MBs, $R_1 = 8.3$ Mbps and $R_2 = 3.6$ Mbps.

Emulation of the R_1/R_2 links. The instantaneous throughput of the BS-to-user and CDN-to-BS links is emulated through application-layer rate control, by counting the number of bits transmitted per second in each HTTP socket and matching them to the output of a 5G channel governed by Rayleigh-distributed fading with $\sigma_1 = R_1 \cdot \sqrt{2/\pi}$ and $\sigma_2 = R_2 \cdot \sqrt{2/\pi}$, respectively.

B. Definition of metrics and methods

The *VLC Output* records the actual video playout as observed by real users: on and off (stall). The *Predicted* output incorporates the timestamps on the actual delivery of packets at the HTTP client to predict the VLC Output. The *Resolution* refers to the DASH logic decision on the requested video bitrate (per segment). The *Cache* records hits and misses at the MEC proxy. The *Rates* metric measures the number of bits delivered per second over the CDN-to-BS (R_2) and BS-to-user (R_1) links (effective throughput). The *MOS* (Stalls) metric assesses the user QoE according to [25]:

$$MOS_{st.} = 3.5 * \exp -(0.15X + 0.19)Y + 1.5, \quad (1)$$

where X is the mean stalling duration and Y is the number of video stalls. X and Y are calculated by using the following methodology. Let T denote the segment playout duration and I the number of video segments that the video player should buffer before it starts playing out the video content at the UE player. In real-life players, I is typically in the range of a few segments (1-2). Let t_i denote the instant (timestamp) where the last bit of segment i is received at the HTTP client and b_i the video playout time of segments (or parts of them) that have been successfully delivered to the HTTP client but haven't been played out yet. We term the last parameter as the *buffered playout time* and we note that by definition $b_1 = 0$ seconds. Accordingly, the initial playout delay is given by t_I and stall events take place whenever $b_i = 0$ for $I \leq i \leq M$, where M denotes the number of segments of the video file. If segment $i-1$ is received before its planned playout time, the buffered playout time for segment i increases by $t_i - t_{i-1}$:

$$b_i = \max\left(0, T + b_{i-1} - (t_i - t_{i-1})\right), I \leq i \leq M. \quad (2)$$

Following a similar approach, we can assess the video stalling time (if any) for segment i as follows:

$$\tau_i = \min\left(0, (t_i - t_{i-1}) - T - b_{i-1}\right), I \leq i \leq M. \quad (3)$$

The mean stalling time is given by $X = \sum_{i=2}^{M} \tau_i / M$ and the number of stalls Y by non-zero elements in $\{\tau_I, ..., \tau_M\}$.

C. Experimental Study Results

In Fig. 2 we assess the performance of *fixed* video streaming over HTTP at 1440p assuming random segment caching and two different proxy strategies (SAG/SWG). To better highlight our key findings, we focus on the first 180 seconds of the video streaming process. Let us focus on plots 1-5 of Fig. 2, which consider the SAG proxy logic. The timing of HTTP client requests per segment ID is marked in the SAG Cache plot (plot 3). In this plot, we observe that a single cache hit (segment 9) during the first 180 seconds. We also observe that random caching over the full set of video bitrates performs poorly, given that the available cache size is $L = 150$MBs and the total file size at 1440p is 418.8 MBs [23]. Thus, a-priori prediction of the target video file and feasible video bitrate per user can alleviate the lack of backhaul resources in the CDN-to-BS link. Nonetheless, as observed by the SAG VLC Output plot (plot 1), if not properly designed, the selection of a fixed video bitrate can be problematic in scenarios where the e2e link cannot support the required network throughput for the target bitrate, i.e., video streaming at 1440p requires roughly 5.28 Mbps for the BBB file considered in our study.

In plot 1, we also observe that VLC awaits to receive two full segments before starting playing out the first segment ($I = 2$) and in some occasions, it chooses not to play out buffered segments immediately, even if their planned playout timing has reached (e.g., segment 3 in plots 1 and 3). This approach enables real-life video players to create a sufficient playout buffer time at the early stage of the video playout and act more aggressively on the selection of higher bitrates for subsequent segments. Due to the requirement to download the full segment at the cellular MEC proxy before starting

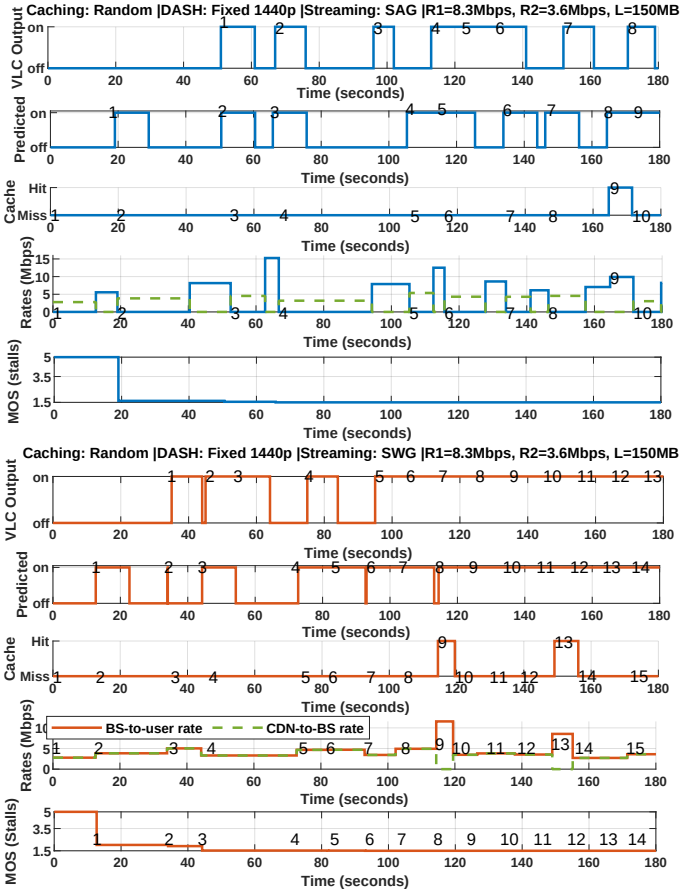


Fig. 2. Fixed resolution @ 1440p with random caching: SAG (plots 1-5) and SWG (plots 6-10). $L = 150$MB, $R_1 = 8.3$ Mbps, $R_2 = 3.6$ Mbps.

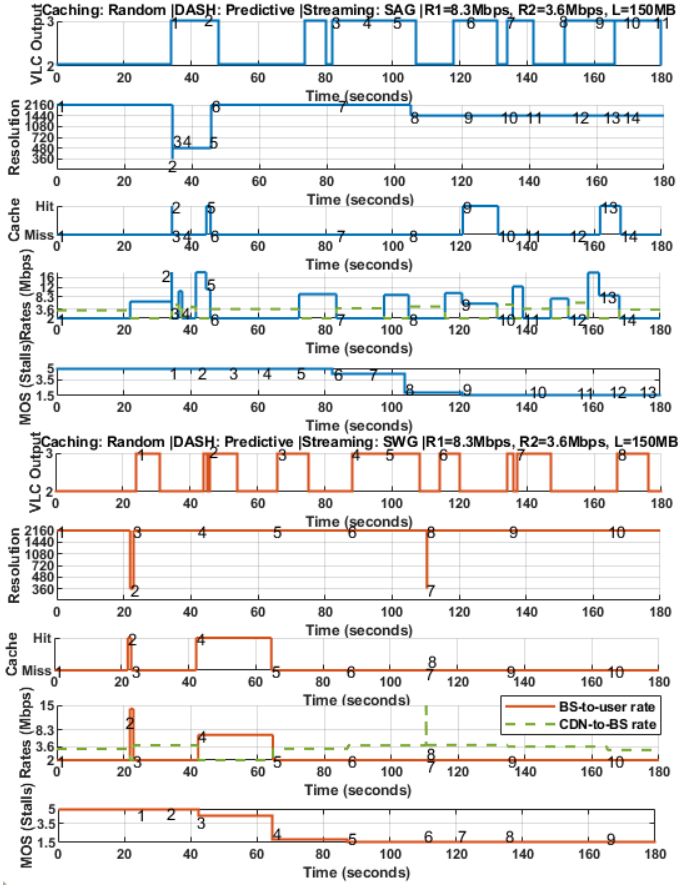


Fig. 3. Predictive DASH with random caching: SAG (1-5) and SWG (6-10). $L = 150$MB, $R_1 = 8.3$ Mbps, $R_2 = 3.6$ Mbps.

delivering it to the UE, SAG inevitably keeps the BS-to-user link idle when receiving the full segment from the CDN-to-BS link. Furthermore, since the VLC streaming logic is limited by segment-by-segment requests, as in most of the existing media players, the CDN-to-BS link remains idle when delivering segments through the BS-to-user link.

The combined effect of cache-agnostic DASH and proxy streaming, increases the effective time-to-content by a factor of $\frac{R_1+R_2}{R_1R_2}$, highlighting that even in its simplest form (i.e., fixed resolution), video streaming over HTTP clashes with network slicing and edge network caching. In the SAG Predicted plot (plot 2), we employ the methodology of section III-B (Eq. 3) for $I = 1$ and observe that even without accounting for the actual value of I, the predicted and actual video playout are very close after a few segments. The SAG MOS plot (plot 5) demonstrates that the user MOS degrades very fast from the early steps of video streaming, due to the large number of video stalls encountered when using SAG.

Let us now focus on the performance of the SWG proxy logic for the same parameter values (plots 6-10 in Fig. 2). In the SWG VLC Output plot (plot 6), we observe a reduced initial playout delay as well as a smoother video playout of segments (as compared to the SAG VLC Output-plot 1), enabling the playout of 5 additional segments within the same observation period. Clearly, the improved performance of SWG is not the result of an enhanced content caching at the MEC-empowered proxy, nor the result of an increased amount of reserved resources. Instead, it is the result of a better utilization of the CDN-to-BS and BS-to-user links as highlighted by SWG Rate plot (plot 9). In plot 9, we observe that SWG fully utilizes the R_2 capacity upon cache miss events, as it enables immediate relaying of smaller HTTP chunks upon reception from the CDN Server. However, the problem of under-utilizing the CDN-to-BS link upon cache hit events (e.g., segments 9 and 13) still remains. Besides, reserved resources in the high-end BS-to-user link are still under-utilized upon cache miss events, matching the BS-to-user to the CDN-to-BS rate performance for the first 8 segments, i.e., this effect is not observed upon cache hit events (9/13) where the full R_1 capacity is exploited. This performance trend highlights the need for incorporating knowledge on the list of cached segments at the UE side and for deploying multi-segment HTTP requests when specific e2e resources are reserved for video streaming over HTTP.

In Fig. 3 we assess the performance of Predictive DASH with random segment caching. In the SAG Resolution plot (plot 2) we observe that Predictive DASH performs aggressively and targets to the highest available resolution from the first segment. After receiving it (32 secs) it initiates the video playout immediately (plot 1) but, given the long delay

experienced (due to cache miss and the use of SAG), it adapts the target resolution for segment 2 to the lowest available (360p). Segment 2 is delivered fast to the HTTP client owing to its small file size and short time-to-content (cache hit). Having no knowledge on the actual reasons that enabled fast reception of segment 2, Predictive DASH targets to a higher resolution for segments 3-5 (480p) and also receives them relatively fast (plot 2). When a cache hit takes place for segment 5, the video player chooses to increase again the requested video bitrate to the highest available (2160p). Predictive DASH chooses to postpone the playout of buffered segments (e.g., 2) until at least one subsequent segment is received (e.g., 3), to smooth out the video playout of subsequent blocks; however, at the cost of temporary stalls (plot 1). After experiencing video stalls for segments 6-7, the DASH logic chooses to lower again the target video resolution to 1440p for segments 8-14. It becomes clear that DASH over cache- and slice-enabled inevitably oscillates the selected video bitrate and severely degrades the video playout performance at the user terminal in the long-term. Oscillation dynamics stem from the cache- and slice-agnostic nature of DASH that cannot robustly infer on the network status, which is actually driven by the segment size, the status of the e2e link and the local cache hits.

The negative impact of oscillation dynamics is also shown in the SAG MOS plot (plot 5), where the user MOS is shown to fast degrade from the early blocks of the playout process. Besides, as shown in the SAG Rate plot (plot 4), adaptive bitrate selection over cache-enabled networks makes it impossible for the underlying network to fully utilize reserved CDN-to-BS and BS-to-user resources, leading to poor social welfare and deteriorated system-wide performance even if network slicing is not used. In the SAG Rate plot (plot 4), we observe that the employment of cache- and slice-agnostic DASH leads to severe under-utilization of both network and storage resources, raising concerns on the robustness of DASH atop 5G/B5G mobile data networks that integrate heterogeneous storage, compute and network resources. Similar trends are observed for the SWG proxy logic in Fig. 3 (plots 6-10). Once again, the reduced time-to-content for specific segments, e.g., segment 2 @360p and 4 @2160p in plot 8, creates oscillations on the video playout (plot 6). Such oscillations are further amplified due the unequal rates reserved for the CDN-to-BS and BS-to-user links. Hence, although the SWG proxy logic enhances the video playout process at the UE (plot 1 vs. plot 6), oscillation dynamics still degrade the user QoE (SWG MOS - plot 10).

IV. Optimal Video Streaming over HTTP in MEC-Empowered Slice-Enabled Networks

Motivated by the study of Section III, we now propose a novel video streaming over HTTP paradigm according to which, the video bitrate should be fixed throughout the video session lifetime in line with i) the slice format parameter values L, R_1, R_2 and ii) the cached content available at the MEC-empowered BS. Such an approach stems from our findings that adaptive bitrate selection and segment-by-segment video requests actually degrade the user QoE in scenarios where specific network resources are dedicated for video streaming. Accordingly, we divide the video streaming process into consecutive *epochs*, with each epoch enabling the UE video player (HTTP client) to batch multiple requests for consecutive video segments of a target video bitrate and better utilize reserved storage/network resources. Full utilization of the CDN-to-BS link capacity dictates caching of the early video segments per epoch, aiming to prolong the time available for fetching non-cached segments through the low-end CDN-to-BS link while delivering both cached and non-cached segments through the high-end BS-to-user link (Fig. 4). Hence, the MEC-empowered proxy BS should i) infer on the maximum video bitrate it can support for the given slice parameter values and video file, ii) identify how segments should be allocated into streaming epochs, and iii) decide which particular segments should be proactively cached for each streaming epoch to mitigate video stalls at the UE side.

A. System Model and Problem Formulation

We consider a MEC-empowered slice-enabled cellular network that provides access to a number of users U, which are interested in consuming popular video files from a tagged content library $\mathcal{F}$. Each video file $f \in \mathcal{F}$ has a playout time of V_f seconds, is encoded at Q_f different bitrates and is partitioned into $M_f = \lceil \frac{V_f}{T_f} \rceil$ video segments, each having a common video playout duration of T_f seconds to support DASH-compatible video streaming. The duration of the last segment is $V_f \mathrm{mod} T_f$ seconds. Let $s_{f,m,q}$ denote the filename and $S_{f,m,q}$ the file size (Mbits) of video segment m that is part of file $f \in \mathcal{F}$ and is encoded at video bitrate q, where $1 \leq m \leq M_f$ and $1 \leq q \leq Q_f$. The MEC-empowered cellular BSs are considered capable of inferring on the list of files $\mathcal{F}_u \subseteq \mathcal{F}$ that each user $u \in \mathcal{U}$ is interested in consuming, e.g., using techniques as in [9]. Based on such predictions, each MEC-empowered BS reserves up to a fixed cache size $L_{u,f}$ for each associated user u to proactively cache video segments of file $f \in \mathcal{F}_u$, potentially at different bitrates.

Without focusing on which particular subset $\mathcal{F}_u$ is selected per user u, or how the maximum cache size $L_{u,f}$ is set by the MEC-empowered cellular BS, i.e., current literature includes numerous works to this end (Section II), we focus on which particular video segments and bitrates should be placed into the MEC-empowered BS's cache for a given file $f \in \mathcal{F}_u$ and user $u \in \mathcal{U}$. In particular, we focus on the emerging scenario where the slice-enabled network can fuel the caching logic of the MEC-empowered BSs with information on specific average throughput guarantees for the CDN-to-BS and the BS-to-user links. Accordingly, each MEC-empowered cellular BS proactively caches popular contents to enhance the video streaming over HTTP process for the attached users, acting as an HTTP proxy that delivers cached video segments directly through the BS-to-user link, or relays non-cached video segments from the CDN to the user through both the CDN-to-BS and BS-to-user links. We further focus on a tagged MEC-empowered BS that serves user $u \in \mathcal{U}$ with file $f \in \mathcal{F}_u$ by reserving resources to guarantee an average throughput of R_1 Mbps for the BS-to-user link and R_2 Mbps for the CDN-to-BS link. The MEC-empowered BS should

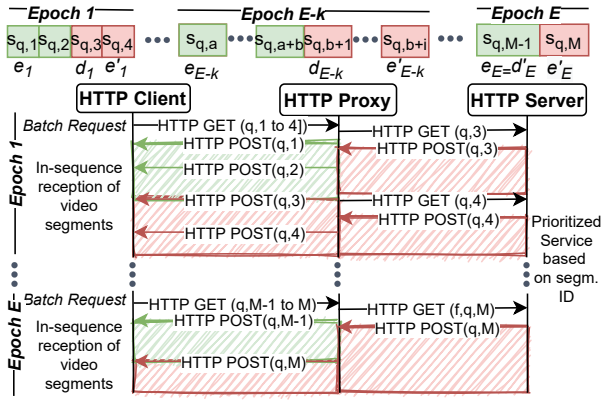


Fig. 4. Epoch-based content caching and video streaming over HTTP

TABLE I
PAPER NOTATION

Notation	Parameter Description
$\mathcal{F}$	Content library (includes all video files)
$\mathcal{U}$	Set of user IDs served by the MEC-empowered network
$\mathcal{F}_u$	Selected files to be cached for a tagged user u
V_f	Video duration of video file f (sec)
T_f	Segment duration of video file f (sec)
M_f	No of video segments for the tagged file f
$s_{f,m,q}$	Filename of segment m encoded at bitrate q for file f
$S_{f,m,q}$	Filesize (Mbits) of segment m encoded at bitrate q for file$f$
$L_{f,u}$	Reserved storage capacity for a user u and file f
R_1	BS-to-user link average throughput guarantee (Mbps)
R_2	CDN-to-BS link average throughput guarantee (Mbps)
σ_1	Rayleigh distribution parameter for BS-to-user link fading
σ_2	Rayleigh distribution parameter for CDN-to-BS link fading
b_i	Aggregate playout time of buffered segments due by the reception of segment i
τ_i	Stalling time experienced by the reception of segment i
I	Minimum no of received segments before video playout
D	Maximum tolerable initial playout video delay (sec)
$q*$	Maximum (optimal) video bitrate assuming zero stalls
E	No of epochs with cached (or not) video segments for file$f$
$[e_k, d_k]$	Sequence of segment IDs cached for epoch k ($\leq E$)
$[d_{k+1}, e'_k]$	Sequence of segment IDs NOT cached for epoch k ($\leq E$)
c^*_q	Caching code indicating the sequence of segment IDs selected for caching (or not) per epoch, for all epochs
C^*_q	Cache size (in Mbits) required to deploy caching code c^*_q

identify the maximum video bitrate q^* that it can support while avoiding video stalls. To this end, it caches segments of video file f taking into account the reserved cache size $L_{u,f}$ and average throughput R_1 and R_2 (Fig. 1).

B. Problem Formulation

Aligned with our discussion in the beginning of Section IV, we consider that the MEC-empowered cellular BS employs epoch-based content caching before the streaming service begins and suggests the most appropriate video bitrate to the end user upon service initiation (SAND). Given the slice $L_{u,f}$, R_1, R_2, and targeting to complete mitigation of video stalls, epoch-based network caching can have more than one feasible solutions. Accordingly, identifying the sequence of cached segments (termed as *caching code*) that minimizes cache usage at the MEC-empowered BS may allow for a better utilization of the available cache, e.g., serving more users, or caching more files. Accordingly, we start our formulation with the problem of epoch-based content caching for a target video bitrate $q \in \{1, ..., Q_f\}$ and then extend it to derive the maximum feasible video bitrate q^* that can be supported.

The MEC-empowered cellular BS should identify the number of epochs E and infer on the consecutive video segments $[e_k, d_k]$ that it should cache per epoch k, where $1 \leq k \leq E$, $e_1 = 1$, $e_k \leq d_k$ and $1 \leq e_k, d_k \leq M_f$. Each epoch should include at least one cached and one non-cached segment ($d_k < e'_k$), i.e., if not, it could have been merged with the subsequent one. Thus, segments e_k to d_k are cached at the MEC-empowered BS while segments $d_{k+1}+1$ to e'_k are fetched by the CDN through the CDN-to-BS and BS-to-user links (see Fig.4). The caching code $[\{e_1, d_1, e'_1\}, ..., \{e_E, d_E, e'_E\}]$, where $e'_k = e_{k+1} - 1$ is the last segment of epoch k and $e_{E+1} = M_f + 1$, should be further constructed so as to respect the slice format $L_{u,f}$, R_1 and R_2 and mitigate video stalls.

We now formulate the problem of epoch-based network caching given a target video bitrate q, file f and user u. Accordingly, we omit subscripts f and u for notational convenience. The optimization problem is to identify the caching code $\mathbf{c}^*_q = [\{e_1, d_1, e'_1\}, ..., \{e_E, d_E, e'_E\}]$ with the minimum cache size requirements C^*_q that accounts for the given slice parameter values and meets the video playout constraints.

Problem Formulation 1. Epoch-based caching for target bitrate q. **Input:** $[L, R_1, R_2, \{S_{m,q}\}, M, T, V, D, q]$

$$C^*_q = \arg_{\{E, e_1, d_1, e'_1, ..., e_E, d_E, e'_E\}} \min \sum_{k=1}^{E} \sum_{m=e_k}^{d_k} S_{m,q} \tag{4}$$

w.r.t.
$$e_k \leq d_k < e'_k < e_{k+1}, \forall k : 1 \leq k \leq E \tag{5}$$
$$e_1 = 1, e'_E = M \text{ and } e_{E+1} = M + 1 \tag{6}$$
$$\frac{S_{1,q}}{D} \leq R_1 \tag{7}$$
$$\sum_{k=1}^{E} \sum_{m=e_k}^{d_k} S_{m,q} \leq L \tag{8}$$
$$\frac{\sum_{m=2}^{l} S_{m,q}}{T \cdot (l-1)} \leq R_1, \forall l : 2 \leq l \leq e'_1 \tag{9}$$
$$\frac{\sum_{m=e_k}^{l_k} S_{m,q}}{T \cdot (l_k - e_k)} \leq R_1, \quad \forall k : 1 < k \leq E \text{ and } \forall l_k : e_k \leq l_k \leq e'_k \tag{10}$$
$$\frac{\sum_{m=d_k+1}^{l_k} S_{m,q}}{T \cdot (l_k - e_k)} \leq R_2, \quad \forall k : 1 \leq k < E \text{ and } \forall l_k : d_k \leq l_k \leq e'_k \tag{11}$$

Eq. (4) derives the (optimal) caching code $\mathbf{c}^*_q = [\{e_1, d_1, e'_1\}, ..., \{e_E, d_E, e'_E\}]$ that minimizes the cache size requirements at the MEC-empowered BS while meeting the constraints in Eqs. (5) to (11). Eq. (5) constrains the caching code to include at least one cached and one non-cached segment per epoch. Eq. (6) bounds the caching code parameters within valid intervals. Eq. (7) bounds the initial playout video delay within a target threshold of D seconds. Eq. (8) constrains the aggregate file size of cached segments not to exceed the available cache size L. Eq. (9) constrains the

aggregate download time of segments 2 to l not to exceed the corresponding video playout time of segment l for all feasible $[2, l]$ intervals, i.e., the delay of receiving the first segment is not a video stall. Eq. (10) acts in a similar fashion for all streaming epochs other than the first one. Eqs. (9) and (10) consider that both cached and non-cached segments are delivered through the BS-to-user link for a tagged epoch. For all streaming epochs $1 \leq k \leq E$, Eq. (11) constrains the download time of non-cached segments ($d_k + 1$ to l_k) to match the video playout time of segment l_k, under all possible $[d_k + 1, l_k]$ intervals ($d_k + 1 \leq l_k \leq e'_k$). In this constraint, we account for the extended time available due to the reception of cached blocks early in the epoch (BS-to-user link).

C. Exhaustive Search Algorithm

Algorithm 1: Identify maximum video bitrate q^* and optimal code $\mathbf{c}^*_{q^*}$ for **Problem Formulation 1**

1 **Input:** $L, R_1, R_2, \{S_{m,q}\}, M, T, V, D$
2 **Output:** $q^*, \mathbf{c}^*_q = \{\{e_1, d_1, e'_1\}, ., \{e_E, d_E, e'_E\}\}, C^*_q$
3 **Function:** `MainEBC` (***Input***)
4 $q^* = -1; C^*_q = -1; \mathbf{c}^*_q = \emptyset$
5 **for** $q = Q$ *to* 1 **do**
6 $c = \sum_{m=1}^{M} S_{m,q}$;
7 **if** $S_{m,q}/D \leq R_1$ **then**
8 $[\mathbf{c}^*_q, C^*_q] = ExhEBC(.., q, 1, M, \emptyset, \{1, M, M\}, c)$
9 **if** $C^*_q \leq L$ && $C^*_q \geq 0$ **then**
10 $q^* = q$; **break;**
11 **end**
12 **end**
13 **end**
14 **return** $q^*, \boldsymbol{c}^*_q, C^*_q$; **end**

Algorithms 1-3 enable us to identify the maximum video bitrate q^* and the optimal solution to Problem Formulation 1. **Algorithm 1** examines all available video bitrates (highest to lowest) and identifies whether a feasible solution exists for a target video bitrate using Algorithm 2. **Algorithm 2** employs exhaustive search to explore the full state space of Problem Formulation 1 and identify the optimal caching code $\mathbf{c}^*_q$ (if any) with the minimum cache size requirements C^*_q. **Algorithm 3** infers if the tagged interval $[e_k, e'_k]$ can meet the constraints of Problem Formulation 1, identifying the segment d_k with the minimum cache requirements.

We now overview **Algorithm 1**. Moving from the highest to the lowest video bitrate $q \in \{1, ..., Q\}$ (step 5), in step 7, we check if the initial playout delay constraint D is met (Eq. (7)). If not, the current target resolution is skipped. If yes, exhaustive search for the segment interval $[1, M]$ is triggered (step 8) assuming full segment caching as baseline. Since Algorithm 2 returns $C^*_q = 0$ if the R_2 capacity is sufficient to deliver all video segments (no caching necessary), or $C^*_q = -1$ if no solution exists (Eqs. (9)-(11) are met), if the cache size constraint of Eq. (8) is met (step 9), the maximum video bitrate q^* is set to the current one (step 10).

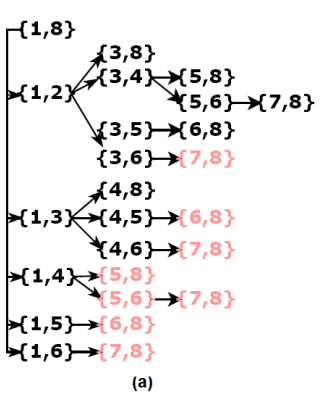
Algorithm 2: Exhaustive search for a target video bitrate q and segment interval $[k_1, k_2]$

1 **Input:** $L, R_1, R_2, \{S_{m,q}\}, M, T, V, D, q, k_1,$
2 $k_2, w, \mathbf{h}_q, H_q$
3 **Output:** $\mathbf{c}^*_q = \{\{e_1, d_1, e'_1\}, ..., \{e_E, d_E, e'_E\}\}, C^*_q$
4 **Function:** `ExhEBC` (***Input***)
5 $[d, cBits] = findSolution(..., k_1, k_2)$
6 **if** $d > 0$ **then**
7 **if** $cost(q, \{S_{m,q}\}, w \cup \{k_1, d, k_2\}) < H_q$ **then**
8 $\mathbf{h}_q = w \cup \{k_1, d, k_2\}$;
9 $H_q = cost(q, \{S_{m,q}\}, w \cup \{k_1, d, k_2\})$;
10 **end**
11 **end**
12 **if** $k_2 - k_1 \geq 2$ **then**
13 **for** $k = (k_1 + 2)$ *to* $(k_2 - 1)$ **do**
14 $[d, cBits] = findSolution(..., k_1, k - 1)$
15 **if** $d > 0$ **then**
16 **if** $cost(q, \{S_{m,q}\}, w \cup \{k_1, d, k - 1\}) < H_q$ **then**
17 $[\mathbf{h}_q, H_q] = recursiveEBC(..., k, k_2, w \cup \{k_1, d, k - 1\}, \mathbf{h}_q, H_q])$;
18 **end**
19 **end**
20 **end**
21 **end**
22 **return** $\boldsymbol{c}_q = \boldsymbol{h}_q, C_q = H_q$;
23 **end**

We now focus on the exhaustive search algorithm (**Algorithm 2**). Parameter w keeps track of the solution search path followed in previous recursions, while parameters $\mathbf{h}_q$-H_q store the optimal caching code-cache size requirements for the interval $[1, k_2]$ in previous calls. Thus, w includes a search path starting from segment 1 and ending up to segment $k_1 - 1$. In step 5, we employ **Algorithm 3** to identify whether a solution exists in $[k_1, k_2]$ to satisfy the rate constraints of Eqs. (9)-(11). If so (steps 6-9), we evaluate if the caching code $w \cup \{k_1, d, k_2\}$ can reduce the cache size requirements as compared to the current optimal code $\mathbf{h}_q$, updating $\mathbf{h}_q$ / H_q if necessary. In steps 12-13, the evaluation of all alternative but valid compositions in $[k_1, k_2]$, containing at least two parts/epochs, is triggered. A valid composition includes epochs composed by at least one cached and one non-cached segment (Eq. (6)). In step 14, we investigate the validity of epoch $[k_1, k - 1]$ and in step 16, we evaluate whether the alternative composition $w \cup \{k_1, d, k - 1\}$ requires a lower cache size as compared to the current solution $\mathbf{h}_q$. If so, a recursion is triggered for the residual interval $[k, k_2]$. This branch-and-bound approach enables fast elimination of solution paths that require more cache, avoiding unnecessary calculations. In steps 12-21, all potential compositions of M are explored and the optimal caching code is identified, i.e., Algorithm 2 is exact. The depth of recursions reaches up to the maximum number of epochs that the segment interval $[k_1, k_2]$ can have, i.e., $E \leq \lfloor \frac{k_2 - k_1 + 1}{2} \rfloor$ (see Appendix A).

Algorithm 3: Validate the rate constraints for segment interval $[e_k, e'_k]$ and identify the optimal d_k

Input: $R_1, R_2, \{S_{m,q}\}, M, T, V, q, e_k, e'_k$
Output: segment d_k, cached bits $C_q[e_k, e'_k]$
Function: `findSolution` (***Input***)
  if $e'_k - e_k < 1$ **then**
    return $d_k = -1, C_q[e_k, e'_k] = -1$;
  end
  $i_1 = e_k; i_2 = e'_k; x = -1; noCache = 1$;
  $cBits = ncBits = totBits = 0$;
  $pTime = T \cdot (e'_k - e_k); iSize = \sum_{m=e_k}^{e'_k} S_{m,q}$;
  if $e_k == 1$ **then**
    $i1 = 2$; $cBits = S_{1,q}$;
  end
  for $m = i_1$ *to* i_2 **do**
    if $totBits + S_{m,q}/(T \cdot (m - i_1 + 1)) > R_1$ **then**
      return $d_k = -1, C_q[e_k, e'_k] = -1$;
    end
    if $totBits + S_{m,q}/(T \cdot (m - i_1 + 1)) > R_2$ **then**
      $noCache = 0$;
    end
    $totBits = totBits + S_{m,q}$;
    if $x < 0$ && $(iSize - totBits - cBits)/pTime \leq R_2$ **then**
      $x = m; cBits = cBits + totBits$;
    end
  end
  if $x == e'_k$ **then**
    return $d_k = -1, C_q[e_k, e'_k] = -1$;
  end
  if $noCache$ **then**
    return $d_k = 0, C_q[e_k, e'_k] = 0$;
  end
  for $m = x + 1$ *to* e'_k **do**
    for $z = m$ *to* e'_k **do**
      if $(ncBits + S_{z,q})/(T \cdot (z - e_k)) > R_2$ **then**
        break;
      end
      $ncBits = ncBits + S_{z,q}$;
    end
    if $z == e'_k$ **then**
      break;
    else
      $cBits = cBits + S_{z,q}$; $ncBits = 0$;
    end
  end
  if $z - 1 == e'_k$ **then**
    return $d_k = -1, C_q[e_k, e'_k] = -1$;
  end
  return $d_k = z - 1, C_q[e_k, e'_k] = cBits$;
end



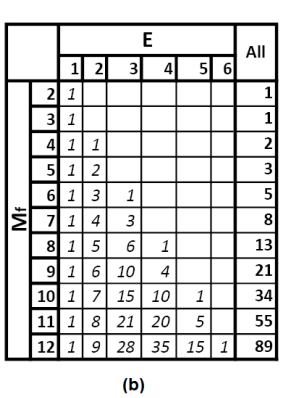


Fig. 5. (a) Run-time example for interval $[1, 8]$, (b) State space size (all) and number of solutions containing exactly E epochs for interval $[1, M]$.

We now overview **Algorithm 3**. Steps 4-6 validate whether the target interval can include at least one cached and one non-cached segment (Eq. (5)). Steps 10-12 are used to validate if the constraint of Eq. (9) is met when $k_1 = 1$. Steps 13-20 validate if the constraints of Eqs. (9)-(10) are met and infer on whether the CDN-to-user link capacity is sufficient to avoid network caching. Steps 21-23 identify the lowest segment ID x that satisfies the rate constraint of Eq. (11) for $l_k = e'_k$. This segment ID can be shifted in steps 31-36, to provide additional time for fetching non-cached segments through the CDN-to-BS link, if complete mitigation of video stalls cannot be achieved. Steps 28-30 are used to infer on whether the reserved CDN-to-BS throughput R_2 can meet the video playout constraints of Problem Formulation 1 without caching. Steps 31-43 validate that all constraints in Eq. (11) are met, shifting the value of x (d_k candidate) to higher values if necessary. Such an event, typically occurs if the file size of early segments in $[x+1, e'_k]$ is comparably larger than those in $[e_k, x]$. Steps 46-47 are executed if Eq. (11) cannot be satisfied.

Fig. 5(a) illustrates how **Algorithm 2** explores the full state space for a video file of $M = 8$ blocks, highlighting with light red the segment intervals that are assessed more than once. In Fig. 5(b), we also highlight the size and structure of the full state space for different sizes M, indicating the number of solution intervals composed by exactly E epochs.

Theorem 1. *The time complexity of Algorithm 1 is upper bounded by* $O(Q \cdot F_{M-1} \cdot M^2)$, *where* $F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$ *is the Fibonacci sequence.*

Proof: See Appendix A.

D. Exact Algorithm of Polynomial Time

We now propose **Algorithm 4** to solve Problem Formulation 1 in polynomial time (replacing Algorithm 2), by keeping record on the outcome of segment intervals that have been previously assessed; thus, mitigating unnecessary recalculations by instantly inferring on the optimal caching code for the respective intervals (and sub-intervals). **Algorithm 4** employs two global MxM arrays to keep track of the cache size requirements and the current optimal caching code per segment interval $[i, j]$ $(1 \leq i \leq j \leq M)$, i.e., **gB** and **gS**, respectively, given the target bitrate q.

Algorithm 4: Proposed DP for a target video bitrate q and segment interval $[k_1, k_2]$

Global: **gS** (MxM array), **gB** (MxM array)
Input: $L, R_1, R_2, \{S_{m,q}\}, M, T, V, D, q, k_1, k_2, w, \mathbf{h}_q, H_q$
Output: $\mathbf{c}_q^* = \{\{e_1, d_1, e_1'\}, ..., \{e_E, d_E, e_E'\}\}, C_q^*$
Function: `ProposedEBC` (***Input***)
  if $\boldsymbol{gB}(k_1, k_2) == 0$ **then**
    $[d, cBits] = findSolution(..., k_1, k_2)$;
    if $d > 0$ **then**
      $\mathbf{gS}(k_1, k_2) = \{k_1, d, k_2\}$;
      $\mathbf{gB}(k_1, k_2) = cost(q, \{S_{m,q}\}, \{k_1, d, k_2\})$;
      if $cost(q, \{S_{m,q}\}, w \cup \{k_1, d, k_2\}) < H_q$ **then**
        $\mathbf{h}_q = w \cup \{k_1, d, k_2\}$;
        $H_q = cost(q, \{S_{m,q}\}, w \cup \{k_1, d, k_2\})$;
      end
    end
    if $k_2 - k_1 \geq 2$ **then**
      for $k = (k_1 + 2)$ *to* $(k_2 - 1)$ **do**
        if $\boldsymbol{gB}(k_1, k-1) == 0$ **then**
          $[d, \mathbf{gB}(k_1, k-1)] = findSolution(..., k_1, k-1)$
          if $d > 0$ **then**
            $\mathbf{gS}(k_1, k-1) = \{k_1, d, k-1\}$;
            $tb = \mathbf{gB}(k_1, k-1)$;
            if $w \neq \emptyset$ **then**
              $tb = tb + \mathbf{gB}(1, k_1 - 1)$;
            end
            if $tb < H_q$ **then**
              $[\mathbf{h}_q, H_q] = ProposedEBC(..., k, k_2, w \cup \{k_1, d, k-1\}, \mathbf{h}_q, H_q])$;
              if $\boldsymbol{gB}(k_1, k-1) > 0$ && $\boldsymbol{gB}(k, k_2) > 0$ && $\boldsymbol{gB}(k_1, k-1) + \boldsymbol{gB}(k, k_2) < \boldsymbol{gB}(k_1, k_2)$ **then**
                $\mathbf{gS}(k_1, k_2) = \mathbf{gS}(k_1, k-1) \cup \mathbf{gS}(k, k_2)$;
                $\mathbf{gB}(k_1, k_2) = \mathbf{gB}(k_1, k-1) + \mathbf{gB}(k, k_2)$;
              end
            end
          else
            $\mathbf{gB}(k_1, k-1) = -1$;
          end
        end
      end
    end
  else if $\boldsymbol{gB}(k_1, k_2) > 0$ **then**
    if $cost(q, \{S_{m,q}\}, w \cup \boldsymbol{gS}(k_1, k_2)) < H_q$ **then**
      $\mathbf{h}_q = w \cup \mathbf{gS}(k_1, k_2)$;
      $H_q = cost(q, \{S_{m,q}\}, w \cup \mathbf{gS}(k_1, k_2))$;
    end
  end
  return $\boldsymbol{c}_q = \boldsymbol{h}_q, C_q = H_q$;
end

We now overview **Algorithm 4**. If the interval of interest $[k_1, k_2]$ has not been examined in previous steps (step 6), we identify the optimal structure for epoch $[k_1, k_2]$ (step 7). If $[k_1, k_2]$ is a valid epoch (step 8), we keep record of the respective epoch structure $\{k_1, d, k_2\}$ and its cache size requirements (steps 9-10), and evaluate whether appending the respective epoch to the caching code path w, which has triggered the call for the segment interval $[k_1, k_2]$, performs better than the current solution $\mathbf{h}_q$ (step 11). If so, we replace the current optimal solution $\mathbf{h}_q$ and update its cache size requirements H_q (steps 12-13). If the interval $[k_1, k_2]$ includes valid sub-intervals (step 16), we trigger the evaluation of all their compositions (step 17). During this evaluation we skip segment intervals that have been previously assessed (step 18) to fully leverage the depth-first search logic employed by the proposed method. If the respective sub-interval $[k_1, k-1]$ has not been assessed before, we identify its optimal structure (step 19), record it (step 21) and store the cache size requirements of the augmented code $w \cup \{\{k_1, d, k-1\}\}$ (steps 22-24).

If the augmented code $w \cup \{k_1, d, k-1\}$ has the potential to perform better than the current solution $\mathbf{h}_q$ (step 26), i.e., branch-and-bound, we append to w the epoch $\{k_1, d, k-1\}$ and trigger a recursive call for the residual interval $[k, k_2]$ (step 27). Accordingly, we evaluate whether the augmented solution of epochs $[k_1, k-1]$ and $[k, k_2]$ performs better than the current optimal caching code in $[k_1, k_2]$ (step 28) and if so, we update the respective records $\mathbf{gS}(k_1, k_2)$ and $\mathbf{gB}(k_1, k_2)$ (steps 29-30). This step builds bottom-up knowledge on the optimal solution and its cache size requirements per interval, taking into consideration all valid compositions of its sub-intervals (due to depth-first search). Step 33 is performed when the interval $[k_1, k-1]$ is not a valid epoch($d \leq 0$), keeping track of this assessment outcome to avoid recalculations (step 34). Step 39 is performed if the interval $[k_1, k_2]$ has been assessed before (disjoint with step 6) and evaluates whether the augmented solution $w \cup \mathbf{gS}(k_1, k_2)$ performs better than the current solution $\mathbf{h}_q$ (step 40). If so, the current optimal solution $\mathbf{h}_q$ is updated with $w \cup \mathbf{gS}(k_1, k_2)$ (steps 41-42).

Proposition 1. *The time complexity and memory usage of Algorithm 4 are bounded by $O(M^4)$ and $O(M^2)$, respectively.*

Proof. See Appendix B.

V. Numerical Results

A. Experimental Setup and Parameters

In this section, we present extensive experimental results according to the system setup and evaluation methodology detailed in Section III-A. For performance comparisons, we consider the following competitive algorithms for the DASH, HTTP Proxy and Cache building blocks of Fig. 1:

Pred-SAG-Random: Adaptive bitrate selection using the Predictive DASH algorithm. HTTP proxy streaming using SAG and random caching over all segments (and bitrates).

Pred-SWG-Random: Similar to *Pred-SWG-Random* but using the SWG (instead of the SAG) HTTP proxy logic.

Fixed-SWG-Random: Fixed resolution according to the output of **Algorithm 1**, assuming the SWG proxy logic and random segment caching over all segments (and bitrates).

Proposed: Fixed resolution according to **Algorithm 1** using the SWG proxy logic and caching based on **Algorithm 4**. Both the HTTP client and the HTTP proxy place multi-segment requests for the full video a-priori, where each request is served in a multi-threaded fashion.

MS-Fixed-SWG-Random: Similar to *Proposed* but assuming random segment caching over all segments (and bitrates).

At this point, we note that multi-segment requests and multi-threaded HTTP service enable the *Proposed* and the *MS-Fixed-SWG-Random* algorithms to send cached video segments parallel to the reception of non-cached video segments (relayed over the CDN-to-BS link); thus, fully leveraging the reserved slice resources in the BS-to-user link. We now define some additional metrics used in our study:

Average Resolution: Averaged over segments played out.

MOS (Resolution): Based on the ITU-T QoE metric under Recommendation P.1203.1 [26]. We consider mode "0" of the P.1203.1 implementation in [27], which assigns a MOS to each segment based on its video bitrate, frame-rate and resolution.

Bitrate Switches: Number of bitrate adaptation events.

Effective R_1' Throughput (Mbps): Total bits sent over the BS-to-user link during the transfer of video segments, averaged over the throughput experienced per segment.

Effective R_2' Throughput (Mbps): Similar to R_1' but with focus on the CDN-to-BS link.

Total Network Usage time (secs): Time duration during which the e2e link is occupied ($<$ actual video playout time).

B. Video streaming performance over time

Fig. 6 extends the study of section III, by evaluating the performance of the *Proposed* and *MS-Fixed-SWG-Random* algorithms under the same system setup of Fig. 3. We start our discussions with the focus on the *MS-Fixed-SWG-Random* algorithm (plots 1-4). In plot 1 (VLC Output), we observe that *MS-Fixed-SWG-Random* benefits from the employment of multi-segment HTTP requests and multi-thread HTTP service, enabling simultaneous download of all cached segments at the MEC-empowered proxy (9, 13, 38, 51); thus, fully leveraging the resources reserved in the BS-to-user link (plot 3 of Fig. 6) while keeping the CDN-to-BS always busy, i.e., in contrast with *Pred-SWG-Random* that leaves the CDN-to-BS link idle upon cache hit events (Section III). Notably, although *MS-Fixed-SWG-Random* fully utilizes the available cache L at the MEC-empowered BS, it is unable to fully leverage reserved resources over the BS-to-user link in the long-term due to its slice-agnostic logic, i.e., the BS-to-user link is not fully utilized after 80 seconds (plot 3) resulting in degraded MOS after playing out a few segments (plot 4).

Moving to the *Proposed* algorithm (plots 5-8), we observe a smooth video playout (VLC Output - plot 5) that attains the highest value for the user MOS (plot 8). By observing the Cache plot (plot 6), we verify that the optimal caching code is given by $h_{f,1440p} = \{1, 26, 64\}$ and that the Proposed algorithm fully leverages multi-segment requests (HTTP client/proxy) as well as multi-thread HTTP service (HTTP proxy/CDN server) for the same parameter values, fetching cached and non-cached video segments simultaneously. By observing plot 7 (Rate plot), we verify that the *Proposed* algorithm fully utilizes both the CDN-to-BS (dashed line) and the BS-to-user links (continuous line), enabling concurrent delivery of all cached segments (1-26) to the UE through the BS-to-user link in parallel to the background delivery of segments 27-36 from the CDN server to the HTTP proxy by second 147. After completing the delivery of cached segments, the BS-to-user link is fully utilized to deliver non-cached video segments that became available through the concurrent CDN-to-BS delivery, i.e., segments 27-31 are delivered at the maximum available BS-to-user rate. This result demonstrates that the Proposed video streaming paradigm not only minimizes the storage requirements at the MEC-empowered BS but also takes full advantage of reserved slice resources to attain the highest feasible video bitrate while completely mitigating video stalls.

C. Resolution- and Stall-Related Metrics

We now turn our attention to the performance of all algorithms for an increasing BS-to-user link capacity R_1. To also assess the impact of a larger CDN-to-BS average link capacity R_2 and available cache size L, we consider the performance of all algorithms for the scenarios: a) $R_2 = 8Mbps$ and $L = 200MBs$ (baseline), b) $R_2 = 8Mbps$ and $L = 50MBs$ (four-fold decrease of cache size) and c) $R_2 = 2Mbps$ and $L = 200MBs$ (four-fold decrease of CDN-to-BS link capacity). Fig. 7(a) highlights that in low to medium R_1 values ($\leq$12Mbps), Predictive DASH algorithms (*Pred-SWG-Random*/*Pred-SAG-Random*) attain a better average resolution performance as compared to the *Proposed*, *MS-Fixed-SWG-Random* and *Fixed-SWG-Random*. Nonetheless, algorithms that adopt the proposed video bitrate selection at fixed resolution attain an enhanced performance when the available link capacity R_1 enables the transfer of large segment sizes, which correspond to the highest available resolution, e.g., $R_1 > 15.8$MBps enables transfer of the largest 2160p segment.

Fig. 7(a) also highlights that the SAG and SWG proxy logic attain a roughly similar performance under the baseline scenario, while the same applies if a four-fold reduction of the available cache size is considered. Nonetheless, a four-fold decrease of the available CDN-to-BS link capacity R_2 from 8 to 2 Mbps, is shown to deteriorate the average resolution performance if *Pred-SAG-Random* is used instead of *Pred-SWG-Random* ($L = 200$Mbps). This performance trend highlights that slice-agnostic DASH has a significantly negative impact compared to cache-agnostic DASH, even if a very large BS-to-user capacity is available.

Interestingly, the results of Fig. 7 reveal that the average resolution performance, which has been widely used in current literature, can be misleading. Taking into consideration that the number of video segments is given by $M = 64$, we observe that *Predictive DASH* algorithms (*SAG* and *SWG*) adapt the video bitrate very frequently (Fig. 7(b)) due to cache hit/miss events, especially when the BS-to-user link capacity R_1 is low. Besides, traditional SAG video streaming is shown to be more vulnerable to oscillation dynamics (Fig. 7(b)) whereas, video streaming at fixed resolution (*Proposed*, *MS-Fixed-SWG-Random* and *Fixed-SWG-Random*), attains the highest

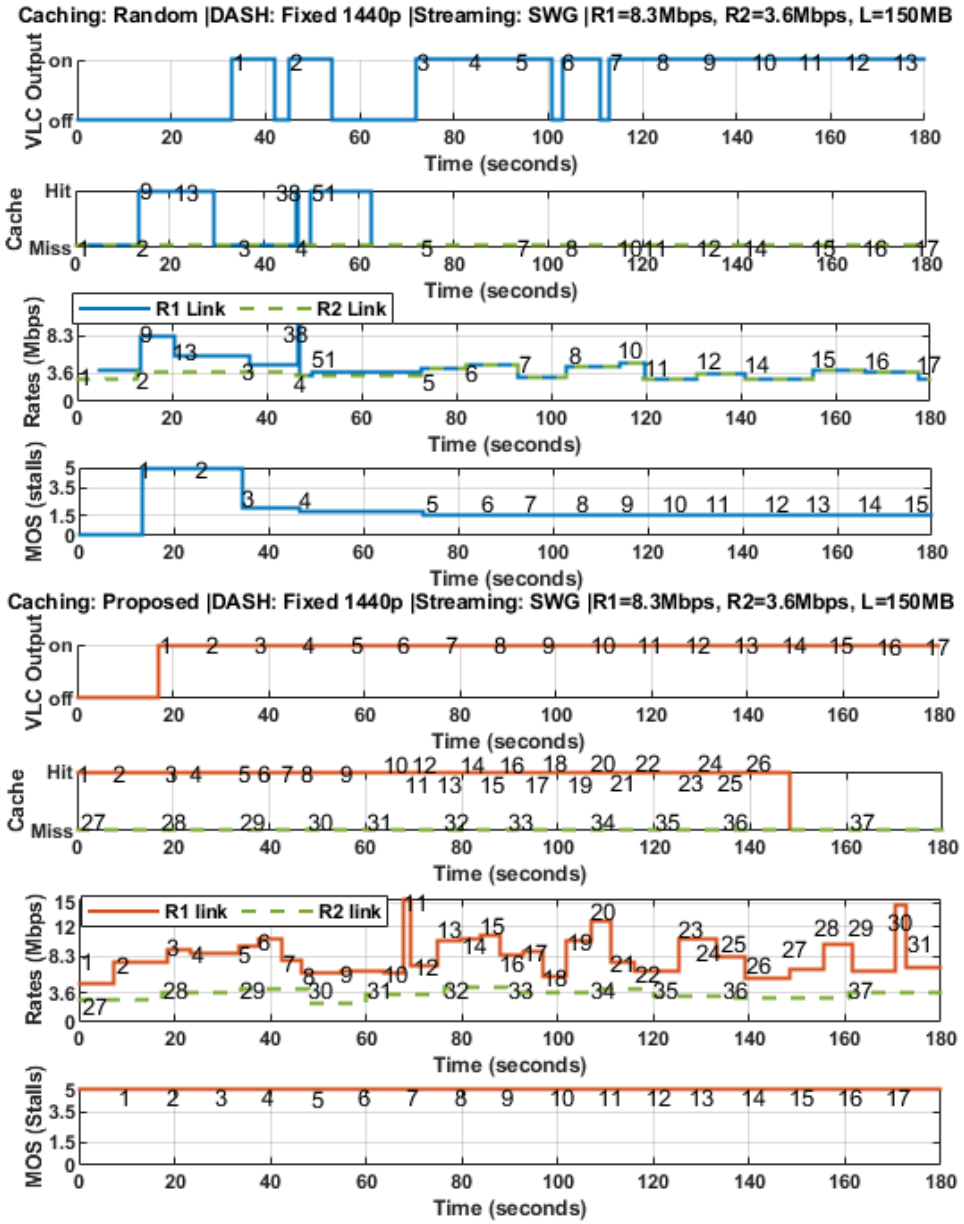


Fig. 6. Video streaming for the *MS-Fixed-SWG-Random* (1-4) and *Proposed* (5-8) algorithms. $L = 150$MB, $R_1 = 8.3$ Mbps, $R_2 = 3.6$ Mbps.

MOS (Fig. 7(c)). This interesting result implies that although *Predictive DASH* algorithms support a higher average resolution performance, they ultimately degrade the user QoE due to very frequent switching across distant resolutions (triggered by cache hit/miss events). This trend is even more evident when the available slice resources enable video streaming at medium to high video resolutions, e.g., for $R_1 > 3$ Mbps where fixed video bitrate selection enables streaming at 720p (Fig. 7(a)).

The *Proposed* algorithm is shown to minimize both the number (Fig. 7(d)) and the duration (Fig. 7(e)) of video stalls, leading to the highest MOS (stallings) performance (Fig. 7(f)), for all parameter values under scope. In Fig. 7, we further observe that although *Pred-SAG-Random* and *Pred-SWG-Random* have similar performance in terms of resolution-related metrics, i.e., (a)-(c) plots, *Pred-SWG-Random* always performs better than *Pred-SAG-Random* in terms of stall-related metrics, i.e., (d)-(f) plots, as it better utilizes the available CDN-to-BS and BS-to-user link capacity. This effect is more evident for a larger cache size L. *MS-Fixed-SWG-Random* and *Fixed-SWG-Random* perform similarly with the *Proposed* algorithm under a large number of system setups, as they are both assumed to employ optimal video bitrate selection (Algorithm 1). Nonetheless, the *Proposed* algorithm attains a superior performance when a small cache size is available (e.g., $L = 50$MBs), due to the fact that the cache hit performance of random caching scales linearly with L. Besides, the *Proposed* algorithm employs optimal epoch-based caching, which enables it to attain better performance when the BS-to-user link capacity R_1 allows the transfer of larger video segments (and bitrates) that require comparably larger cache

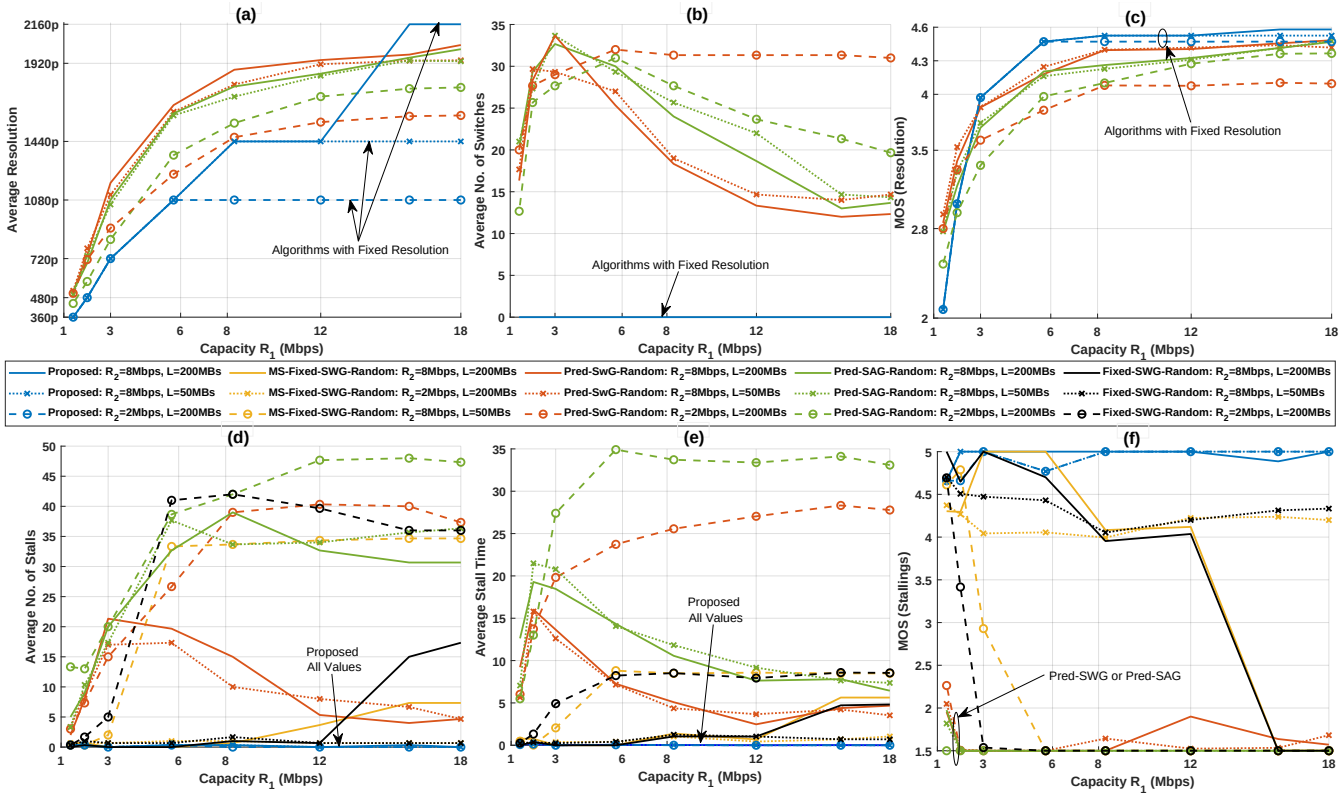


Fig. 7. Performance comparisons of all algorithms vs. maximum average capacity threshold R_1: Resolution- and Stall-related metrics

size. For example, *MS-Fixed-SWG-Random* and *Fixed-SWG-Random* are shown to encounter a large amount of video stalls (Fig. 7(d)/(e)) and degrade the MOS (stallings) performance (Fig. 7(f)) for $R_1 > 15$Mbps. Besides, the *Proposed* algorithm attains at least the same performance with them by utilizing comparably lower cache size (see section V-D).

D. Network and Cache Usage Metrics

Fig. 8 plots the performance of all algorithms under network and cache usage metrics. Starting with the scenario where limited backhaul resources are reserved, i.e., $R_2 = 2Mbps$, we observe that *Pred-SAG-Random* necessitates a comparably larger BS-to-user capacity R_1, i.e., in lower R_1 values the effective R_1' and R_2' throughput does not match the reserved e2e capacity of $min(R_1, R_2)$. On the other hand, *MS-Fixed-SWG-Random* exhibits similar performance with *Fixed-SWG-Random* and *Pred-SWG-Random*, i.e., matching $min(R_1, R_2)$, indicating that the capability to simultaneously request/receive multiple segments cannot be fully harnessed when random caching is employed if the available R_2 link is limited. In the contrary, the *Proposed* algorithm scales better its effective BS-to-user throughput R_1' to the reserved R_1 link capacity, due to optimal epoch-based caching, e.g., when $R_2 = 2Mbps$ and $R_1 < 6Mbps$ in Fig. 8(a), the *Proposed* algorithm almost doubles the effective throughput R_1'. Nevertheless, this performance trend reaches to a point where an increase of the BS-to-user link capacity R_1 cannot be further harnessed.

In particular, the *Proposed* algorithm selects a target video bitrate only if the transfer of its largest video segment can be achieved by completely mitigating video stalls; thus, since the largest video segment can be comparably larger than the average segment size, the effective R_1' throughput can be lower than the average bitrate, especially at the end of caching epochs. For that reason, we observe that the *Proposed* algorithm requires a comparably lower network usage time when $R_2 = 2Mbps$ (Fig. 8(c)), indicating that the *Proposed* algorithm releases reserved network and storage resources earlier than the actual video playback time; thus, providing more space for service to other network applications. In the contrary, for $R_2 = 2Mbps$, we observe that all competitive (other than the *Proposed* one) algorithms require a significantly larger network usage time to complete the delivery of video segments (Fig. 8(c)), higher than the playback time (Figs. 7(d)/(e)).

In the scenarios where backhaul capacity is $R_2 = 8Mbps$, the *Proposed* and *MS-Fixed-SWG-Random* algorithms are shown to enhance both the effective BS-to-user R_1' (Fig. 8(a)) and the CDN-to-BS R_2' throughput (Fig. 8(b)) as compared to the *Pred-SWG-Random* algorithm, due to their native capability to request and receive multiple video segments in parallel. Besides, for that reason, the *Proposed* and *MS-Fixed-SWG-Random* algorithms are shown to linearly scale their effective throughput performance (R_1' and R_2') in values lower than $min(R_1, R_2)$. Nevertheless, due to the employment of optimal epoch-based caching, the *Proposed* algorithm attains a superior performance compared to *MS-Fixed-SWG-Random* in scenarios where R_1 is large enough to support the transfer of large file sizes (2160p), e.g., $R_1 > 15.8$. The performance gains following from optimal epoch-based caching are also

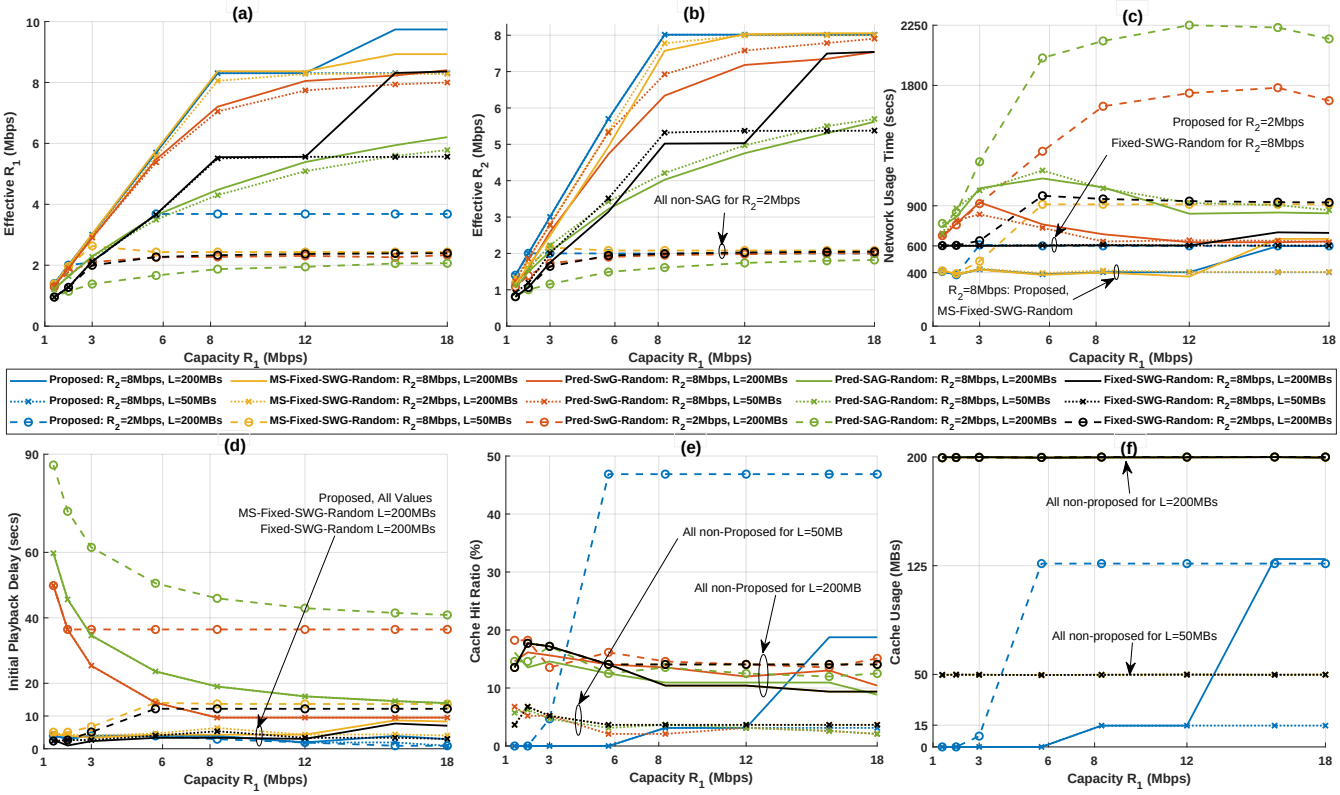


Fig. 8. Performance comparisons of all algorithms vs. maximum average capacity threshold R_1: Network / Cache usage metrics

shown in Fig. 8(d), where the *Proposed* algorithm outperforms all competitive algorithms in terms of Initial Playback delay, due to the fact that the *Proposed* algorithm a) always caches the first video segment when $R_2 < R_1$, and b) employs joint optimization of video bitrate selection and segment caching. Fig. 8(d) illustrates that cache- and slice-agnostic *Predictive DASH* (SAG or SWG), results in comparably larger initial playback delay, especially when the e2e link capacity is low.

Fig. 8(e) plots the cache hit ratio (CHR) performance. As expected, all algorithms employing slice-agnostic random caching attain a roughly similar performance for the same cache size L. On the contrary, the *Proposed* algorithm is shown to adapt its CHR performance depending on the available R_1 and R_2 capacity guarantees. For example, when $R_2 = 2Mbps$, the *Proposed* algorithm attains a very high CHR to alleviate the scarcity of backhaul resources whereas, for $R_2 = 8Mbps$, it attains a low CHR performance for $R_1 < 15.8Mbps$ and a high CHR for $R_1 \geq 15.8Mbps$ to sustain the highest available resolution (Fig. 7(a)). This performance trend highlights that maximizing the CHR is not necessarily a key requirement for MEC-empowered slice-enabled 5G/B5G data networks, which aim to guarantee some minimum QoS/QoE through the joint utilization of reserved network and cache resources. Besides, as compared to all competitive solutions, the *Proposed* algorithm attains a superior QoE performance (based either on resolution, or video stalls) while minimizing the cache usage at the MEC-empowered BS (Fig. 8(f)); thus, releasing a considerable amount of storage and network resources for other services, or users.

E. Run time Performance

We now measure and compare the run time performance for deriving the optimal solution using either the Exhaustive search (Algorithm 2), or the Proposed DP algorithm (Algorithm 4) in Fig. 9. Recall that the time complexity of both algorithms scales with the number of video segments M (Theorem 1/Proposition 1). By assuming the same slice parameters with Fig. 6, we focus on the run time performance under four different segment size distributions: a) segments of the BBB video at 1440p resolution (BBB@1440p), b) equal-sized segments with size equal to the average segment size of the BBB@1440p video, c) normally-distributed segment sizes assuming the average segment size and standard deviation of the BBB video@1440p, and d) randomly-distributed segment size assuming that the average segment size equals that of the BBB video@1440p. As expected, the run time of exhaustive search scales exponentially with M, whereas the proposed algorithm requires polynomial run time. Besides, the run time of the proposed algorithm is in the order of a few seconds for all parameters under scope, whereas the run time of exhaustive search scales exponentially with M, e.g., for $M = 60$, the proposed algorithm requires 3 seconds to calculate the optimal solution, whereas the exhaustive search requires about 1.5 day (128.540 secs) to reach the same output. This observation clearly highlights the importance of our contributions, including the proposed system modeling and solution. It is also interesting to note that the size distribution of video segments has a non-negligible impact on the runtime performance of both algorithms, mainly due to the fact that the branch-and-

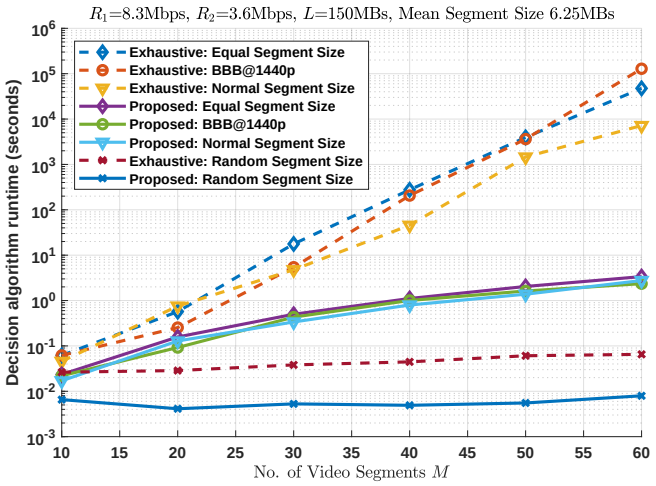


Fig. 9. Run time performance for exhaustive search and proposed ECB

bound approach adapts to the structure/values of the solution space and does not employ a fixed number of steps.

In particular, both algorithms exhibit the highest run time requirements for the equal-sized scenario, which in general can be considered as a trivial solution scenario. This performance trend is somewhat expected, provided that equal-sized segments trigger full exploration of the solution state space (for both algorithms). The run time of each algorithm in the BBB@1440p and normal segment size scenarios is lower, yet not very far away from that of the equal-sized segment scenario. On the other hand, both algorithms seem to infer on the optimal solution at a sub-sec scale if randomly-sized segments constitute the target file. This performance trend readily follows from the fact that branch-and-bound enables fast elimination of solution paths that either require more cache, or fail to meet the optimization constraints.

VI. Conclusion

In this paper, we have provided a comprehensive study to highlight that traditional video streaming over HTTP using DASH and/or SAG HTTP proxies cannot harness the potential of MEC and network slicing in 5G/B5G networks. Accordingly, we have proposed a novel video streaming over HTTP paradigm that exploits information on the available cache size, the file size of video segments as well as knowledge of the reserved BS-to-user and CDN-to-BS network resources, to identify the maximum attainable video bitrate that guarantees complete mitigation of video stalls. An optimal algorithm for solving the joint problem of video bitrate selection and epoch-based caching has been proposed, while its time/memory complexity have been assessed. Through extensive numerical results, we have demonstrated that the proposed video streaming over HTTP algorithm outperforms traditional and contemporary DASH solutions under a vast set of performance metrics, enabling better utilization of reserved network/cache resources, full mitigation of video stalls and maximized user QoE. As future work, we aim to develop low-complexity variants of the proposed polynomial-time yet exact algorithm, investigating the trade-off between time complexity and performance. Future work also includes extension of our system model and proposed solution in view of multi-source video streaming scenarios.

Appendix A
Proof of Theorem 1

The time complexity of **Algorithm 3**, which identifies if a target segment interval $[e_k, e'_k]$ can be considered as a valid epoch, is given by $n_k + \sum_{m=x_k}(n_k - m) + C$, where $n_k = e'_k - e_k + 1$ is the number of segments in $[e_k, e'_k]$, C is a number of constant evaluation steps, and x_k takes values according to steps 21-23. Considering that $1 \leq x_k \leq M$, it readily follows that the time complexity of Algorithm 3 is bounded by $O(n_k + n_k \cdot (n_k - 1)/2 + C)$, which is $O(n_k^2)$. Moreover, the time complexity required to identify if a given caching code $c = \{\{1, d_1, e'_1, \}, ..., \{e_E, d_E, e'E\}\}$ that is composed by exactly E epochs is valid, i.e., if c satisfies Eqs. (5)-(11), is given by $O(\sum_{k=1}^{E}(e'_k - e_k + 1)^2)$. This operation is performed by **Algorithm 2** in a recursive fashion, by employing Algorithm 3 for each one of the different compositions of M with E parts on an epoch-by-epoch basis. Taking into account that $\sum_i (n_i)^2 < (\sum_i n_i)^2$ for $n_i > 0$ and that $\sum_{k=1}^{E}(e'_k - e_k + 1) = e'_E - e_1 + 1 = M$, the time complexity required to infer on the feasibility of a tagged caching code c, which is composed by exactly E epochs, is upper bounded by $O(M^2)$, independent of the sequence of epoch intervals $\{e_k, e'_k\}$ ($k \in \{1, ..., E\}$).

Considering that there exists up to one caching code to minimize the cache requirements of a tagged segment interval (**Algorithm 3** derives this code), it follows that the number of all valid caching codes that the exhaustive search algorithm should explore is equal to the number of compositions of M into E parts with size greater than one, i.e., a valid epoch should include at least one cached and one non-cached segment. Besides, for that reason, the number of epochs E ranges from 1 to $\lfloor \frac{n}{2} \rfloor$. We observe that each composition of M in E parts is equivalent with the composition of $M+E$ into E parts with each part having size greater than 1 (i.e., we increase each part by 1). Thus, since the number of compositions of M into E parts is given by the binomial $\binom{E-1}{M-1}$, the compositions of M into E parts of size greater than one, is given by the composition of $M - E$ into E parts, i.e., $\binom{E-1}{M-E-1}$. Accordingly, when the full segment interval $[1, M]$ is considered, the time complexity of Algorithm 2 is bounded by $O\left(\sum_{k=1}^{\lfloor \frac{M}{2} \rfloor} M^2 \cdot \binom{E-1}{M-E-1}\right) = O(M^2 \cdot F_{M-1})$, where F_n is the Fibonacci sequence. Since **Algorithm 1** triggers a call to **Algorithm 2** for up to Q times, we reach to Theorem 1.

Appendix B
Proof of Proposition 1

Algorithm 4 keeps track of previous visits to segment intervals and eliminates recalculations on their optimal caching code (and validity) in future steps. Due to its depth-first search logic, stepping out from a given interval $[i, j]$ takes place only if all possible solutions in sub-branches (sub-intervals) have been evaluated (i.e., steps 17 and 27). Accordingly, **Algorithm 4** visits every segment interval $[i, j]$ only once to i) infer

on its validity as an epoch and identify its optimal structure in $O((j-i+1)^2)$ using **Algorithm 3**, and ii) perform a number of constant steps, e.g., to evaluate augmented solutions in steps 39-42, or steps 29-30. Hence, the time complexity of **Algorithm 4** is derived as the sum of unique segment intervals of size greater than one multiplied by the time complexity required to identify the optimal solution for each such interval. The time complexity of **Algorithm 3** is derived in Appendix A. Thus, we now assess the number of unique segment intervals ($j-i \geq 2$) in $[1, M]$ of a tagged size $k \geq 2$. A valid caching code should include segment intervals of size greater than one; thus, any interval $[i, j]$ starting with $i = 2$, or ending to $j = M-1$, is not valid and should be excluded from the evaluation (steps 17-18). For $M > 3$, we observe that i) there exists exactly one segment interval of size $k = M$, i.e., $[1, M]$, ii) there exist no valid intervals of size $k = M-1$, iii) there exist two segment intervals of size $k = M-2$, i.e., $[1, M-2]$ and $[3, M]$, and iv) there exist exactly $M-k-1$ valid intervals of size k for all $k \in \{2, ..., M-3\}$. Accordingly, the number of valid and unique intervals in $[1, M]$ is given by $3 + \sum_{k=2}^{M-3}(M-k-1) = \frac{1}{2}M(M-5)+5$. Hence, the time complexity of Algorithm 4 is derived by:

$$\begin{aligned} & O\left(M^2 + 2\cdot(M-2)^2 + \sum_{k=2}^{M-3} k^2\cdot(M-k-1)\right) \\ = \; & O\left(\frac{M^4}{12} - \frac{M^3}{3} + \frac{29M^2}{12} - \frac{31M}{6} + 6\right), \end{aligned} \tag{12}$$

which is $O(M^4)$. The memory complexity $O(M^2)$ comes from the requirement of keeping track previously visited segment intervals using two MxM arrays, i.e., **gS** and **gB**.

ACKNOWLEDGMENT

The author would like to acknowledge the valuable contribution of Mr. Nikolaos Episkopos to the coding and derivation of experimental measurements presented in this work.

REFERENCES

[1] C. Bernardos et al., "European Vision for the 6G Network Ecosystem",The 5G Infrastructure Association (5GIA), July 2021.
[2] S. Kekki et al., "MEC in 5G networks", 1st ed., Eur. Telecommun. Standards Inst., ETSI White Paper 28, Jun. 2018.
[3] ETSI MEC, "Multi-access Edge Computing (MEC); Use Cases and Requirements", ETSI GS MEC 002 V3.1.1, Apr. 2023.
[4] P. Cerwall et al., "Ericsson Mobility Report", *White Paper*, Nov. 2021.
[5] A. A. Barakabitze et al., "QoE Management of Multimedia Streaming Services in Future Networks: A Tutorial and Survey", *IEEE Commun. Surv. & Tut.*, vol. 22, no. 1, pp. 526-565, Q1 2020.
[6] A. Bentaleb et al., "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP", *IEEE Commun. Surv. & Tut.*, vol. 21, no. 1, pp. 562-585, Q1 2019.
[7] 3GPP TS 38.300 V16.8.0, "NR; NR and NG-RAN Overall Description; Stage-2", *3GPP*, Technical Specification, Dec. 2021.
[8] D. King et al., "FG-NET2030 – Focus Group on Technologies for Network 2030, Network 2030 - Terms and Definitions for Network 2030", *Standard. Sector of ITU*, Technical Specification, Jun 2020.
[9] H. S. Goian et al., "Popularity-Based Video Caching Techniques for Cache-Enabled Networks: A Survey", *IEEE Access*, vol. 7, pp. 27699-27719, March 2019.
[10] T. M. Ayenew, D. Xenakis, N. Passas and L. Merakos, "Cooperative Content Caching in MEC-Enabled Heterogeneous Cellular Networks", *IEEE Access*, vol. 9, pp. 98883-98903, July 2021.
[11] D. Jiang et al., "Analysis and Optimization of Caching and Multicasting for Multi-Quality Videos in Large-Scale Wireless Networks", *IEEE Trans. on Comm.*, vol.67, no.7, pp.4913-4927, Jul 2019
[12] W. Li et al., "Quality of Experience in ICN: Keep Your Low- Bitrate Close and High-Bitrate Closer", *IEEE/ACM Trans. on Netw.*, vol. 29, no. 2, pp. 557-570, April 2021.
[13] B. Wang et al., "Improving the Performance of Online Bitrate Adaptation with Multi-Step Prediction Over Cellular Networks", *IEEE Trans. on Mobil. Comput.*, vol.20, no.1, pp.174-187, 1 Jan 2021
[14] S. Yang et al.,"Multi-Access Edge Computing Enhanced Video Streaming: Proof-of-Concept Implementation and Prediction/QoE Models", *IEEE Trans. on Vehic. Techn.*, vol.68, no.2, pp.1888-1902, Feb 2019
[15] G. S. Park et al., "Video Quality-Aware Traffic Offloading System for Video Streaming Services Over 5G Networks With Dual Connectivity", *IEEE Trans. on Vehic. Techn.*, vol.68, no.6, pp.5928-5943, Jun 2019
[16] A. Mehrabi, M. Siekkinen and A. Ylä-Jääski, "Edge Computing Assisted Adaptive Mobile Video Streaming", *IEEE Trans. on Mob. Comput.*, vol. 18, no. 4, pp. 787-800, 1 April 2019.
[17] X. Huang et al., "Towards 5G: Joint Optimization of Video Segment Caching, Transcoding and Resource Allocation for Adaptive Video Streaming in a Multi-Access Edge Computing Network", *IEEE Trans. on Vehic. Techn.*, vol.70, no.10, pp.10909-10924, Oct 2021
[18] T. X. Tran et al., "Adaptive Bitrate Video Caching and Processing in Mobile-Edge Computing Networks", *IEEE Trans. on Mobil. Comput.*, vol.18, no.9, pp.1965-1978, Sept 2019.
[19] RE-CENT video streaming toolbox, GitHub. [Online]: https://github.com/Fogus-Gr/recent-dash-proposed-caching
[20] *Big Buck Bunny*, short computer-animated comedy film featuring animals of the forest, made by the Blender Institute, part of the Blender Foundation, 2008. [Online] peach.blender.org.
[21] *FFmpeg*, suite of libraries and programs for handling video, audio, and other multimedia files and streams. [Online]: ffmpeg.org
[22] J. Le Feuvre, "GPAC filters", Proceedings of the 11th ACM Multimedia Systems Conference, pp. 249-254, May 2020.
[23] N. Episkopos, D. Xenakis, "Big Buck Bunny Segments", GitHub. [Online]: https://gain.di.uoa.gr/DASH/dash_segments_bbb.7z
[24] VideoLan, "VLC media player". [Online] videolan.org/vlc/.
[25] T. Hoßfeld et al., " Internet Video Delivery in YouTube: From Traffic Measurements to Quality of Experience", in Data Traffic Monitoring and Analysis, Lecture Notes in Computer Science, Springer, vol 7754, 2013.
[26] ITU-T Series P: Telephone transmission quality, telephone installations, local line networks, "Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport - Video quality estimation module", Recommendation ITU-T P.1203.1, Jan. 2019.
[27] S. Göring et al., "ITU-T Rec. P.1203 Standalone Implementation", July 2021. [Online]: https://github.com/itu-p1203/itu-p1203.

Dr. Dionysis Xenakis is Assistant Professor at the Department of Digital Industry Technologies of the National and Kapodistrian University of Athens (NKUA), Greece. Dionysis received the Ph.D. degree from the Department of Informatics and Telecommunications at NKUA in 2014 and has participated in numerous EU-funded projects, serving as Project Coordinator, Technical Manager, PI and Researcher. He was recipient of the prestigious Onassis PhD and Postdoc fellowships. Dionysis has co-authored more than 40 peer-reviewed journal and conference papers, while he has chaired and served as TPC member in numerous top-tier IEEE conferences (IEEE GLOBECOM, IEEE ICC, etc.). Dionysis has been reviewer to almost all high-ranking IEEE journals in Computer Science - Data Networks and is currently Editor in IEEE Networking Letters. He has also served as Selection Member Committee in the 2021/2022/2023 IEEE ComSoc Student Competition, representing the EMEA region. Dionysis is head of the CISCO Networking Academy at NKUA Greece and member of Industry-Academia Committees promoting critical synergies between the two sectors. His current research interests lie in the design and analysis of MEC-enabled 5G/6G mobile data networks, Distributed Ledger Technologies and Digital Twins.