CD-LwTE: Cost- and Delay-aware Light-weight Transcoding at the Edge

Alireza Erfanian, *Student Member, IEEE*, Hadi Amirpour, *Member, IEEE*, Farzad Tashtarian, *Member, IEEE*, Christian Timmerer, *Senior Member, IEEE*, and Hermann Hellwagner, *Senior Member, IEEE*

***Abstract*—The edge computing paradigm brings cloud capabilities close to the clients. Leveraging the edge's capabilities can improve video streaming services by employing the storage capacity and processing power at the edge for caching and transcoding tasks, respectively, resulting in video streaming services with higher quality and lower latency. In this paper, we propose *CD-LwTE*, a *C*ost- and *D*elay-aware *L*ight-*w*eight *T*ranscoding approach at the *E*dge, in the context of HTTP Adaptive Streaming (HAS). The encoding of a video segment requires computationally intensive search processes. The main idea of *CD-LwTE* is to store the optimal search results as metadata for each bitrate of video segments and reuse it at the edge servers to reduce the required time and computational resources for transcoding. Aiming at minimizing the cost and delay of Video-on-Demand (VoD) services, we formulate the problem of selecting an optimal policy for serving segment requests at the edge server, including (*i*) storing at the edge server, (*ii*) transcoding from a higher bitrate at the edge server, and (*iii*) fetching from the origin or a CDN server, as a Binary Linear Programming (BLP) model. As a result, *CD-LwTE* stores the popular video segments at the edge and serves the unpopular ones by transcoding using metadata or fetching from the origin/CDN server. In this way, in addition to the significant reduction in bandwidth and storage costs, the transcoding time of a requested segment is remarkably decreased by utilizing its corresponding metadata. Moreover, we prove the proposed BLP model is an NP-hard problem and propose two heuristic algorithms to mitigate the time complexity of *CD-LwTE*. We investigate the performance of *CD-LwTE* in comprehensive scenarios with various video contents, encoding software, encoding settings, and available resources at the edge. The experimental results show that our approach (*i*) reduces the transcoding time by up to 97%, (*ii*) decreases the streaming cost, including storage, computation, and bandwidth costs, by up to 75%, and (*iii*) reduces delay by up to 48% compared to state-of-the-art approaches.**

***Index Terms*—Video streaming, transcoding, video on demand, edge computing, network function virtualization (NFV).**

I. Introduction

A. Motivation

In HAS, a video is divided into short intervals known as video segments, and they are provided at multiple representations. Different representations embody different quality levels of a segment, potentially at different spatial resolutions, and are encoded at different bitrates. As for our approach, the bitrate is the most relevant characteristic of a representation, we will mostly use the term *bitrate* to denote a representation throughout this paper. Since a video is offered in different bitrates (representations) for each segment, a client can choose for each segment the most compatible representation according to the device properties and currently available network bandwidth and, thus, adapt the video playout to dynamically changing network conditions.

The edge computing paradigm brings storage and computation capabilities close to the clients, enabling more efficient video streaming. Leveraging the storage capacity at the edge allows for caching the popular segments/bitrates at the edge to serve clients' requests with minimum latency. Although storage devices come with enormous capacities and lower prices nowadays, considering the massive amount of videos and storage limitations on the edge servers, it is impossible to store all segments/bitrates at an edge server. Moreover, it incurs high overhead in storage to keep all segments/bitrates, especially for those video segments/bitrates that are rarely requested. Cha *et al.* [1] showed that video requests follow a long-tail access pattern, and only a small fraction of videos are popular. Popular videos account for almost 80% of the total views, while other videos receive few requests. Considering this fact, some papers like [2]–[7] proposed transcoding-enabled adaptive video streaming by storing the popular segments/bitrates at the edge server. They store only the highest bitrate of unpopular segments and prepare the remaining bitrates through transcoding processes. In other words, when a client requests a popular segment/bitrate, it will be served immediately from the edge's storage, which imposes *storage cost*. In contrast, a request for an unpopular segment/bitrate will be served by transcoding from a higher bitrate to the desired one, which results in *computation cost*. However, transcoding is inherently a computationally-intensive and time-consuming process, imposing significant cost and delay.

B. Challenges and Contributions

In our previous work [5], we intended to answer the following questions: *(i) How to design a light-weight transcoding method at the edge? (ii) How the proposed transcoding method can be cost-effective at the edge?* For the first research question, we introduced a novel technique for transcoding called *L*ight-*w*eight *T*ranscoding at the *E*dge (*LwTE*), motivated by the aforementioned issues. In the direction of the second research question, in [5], we formulated the problem of selecting a policy, *i.e.*, store and transcode, for each segment/bitrate under unlimited available resources at the edge. The results proved the *LwTE*'s feasibility and cost-efficiency

The authors are with Christian Doppler Laboratory ATHENA, Alpen-Adria-Universität, Klagenfurt, Austria; email: alireza.erfanian@aau.at. The first two authors contributed equally to this work.

compared to the conventional and state-of-the-art methods for simple, yet practical scenarios. However, it suffers from the following issues: *(i)* considering unrealistic assumptions like unlimited available resources at the edge server; *(ii)* ignoring the impact of serving delay; *(iii)* insufficient investigation of the *LwTE*'s performance from the encoding aspect; *(iv)* the lack of comparison with various video compression standards.

To address the above issues, in this study, we extend our investigation in [5] by proposing *C*ost- and *D*elay-aware *L*ight-*w*eight *T*ranscoding at the *E*dge (*CD-LwTE*) as a more comprehensive and realistic approach. To evaluate *CD-LwTE*'s performance in a more realistic situation, we relax some assumptions of [5] by adding resource constraints at the edge and considering a new policy, *i.e.*, the *fetch* policy, for serving the requests at the edge. In the same direction, we also *add serving delay* to the objective of selecting a policy for each segment/bitrate, aiming to minimize the total cost and serving delay. From the encoding aspect, we extend the method of extracting metadata by adding more optimal decisions, such as the TU partitioning structure and motion vectors, to the metadata to further decrease transcoding time at the edge. Moreover, we extend our experiments on the encoding aspect of *LwTE* by considering video content complexity, segment duration, encoding software, encoding setting, and processing power. Therefore, the main contributions of this paper are as follows:

- We propose a framework to serve the clients' requests at the edge server with *limited resources* by selecting a suitable policy including $store$, $transcode$, and $fetch$ for each segment/bitrate, and react to changing clients' request patterns and available resources at the edge server.
- We formulate the problem of minimizing the total cost, including storage, computation, and bandwidth costs, and requests' serving delay as a Binary Linear Programming (BLP) model and prove its NP-hardness.
- To mitigate the time complexity of the proposed BLP model, we introduce two heuristic algorithms of polynomial time complexity that achieve close performance to the BLP one in our simulations.
- We add more optimal decisions to the metadata to further decrease the transcoding time at the edge. The evaluation results indicate that the transcoding time is decreased by up to 97%.
- We investigate various *content-dependent* parameters like video content complexity and segment length on *CD-LwTE*'s performance.
- We compare the performance with conventional transcoding (without using metadata) using both AVC/H.264 and HEVC/H.265 video compression standards in various encoding settings and processing power profiles.
- We compare *CD-LwTE* with the state-of-the-art approaches. The results indicate that *CD-LwTE* compared with the state-of-the-art approaches reduces the streaming cost and delay by up to 75% and 48%, respectively.

C. Paper Organization

The remainder of the paper is organized as follows. Section II highlights related work. *CD-LwTE* is described in Section III and evaluated in Section IV. Section V concludes the paper and outlines future work.

II. Related Work

Leveraging edge capabilities to deliver videos efficiently is widely used in video streaming applications. We classify these approaches into the following categories based on the employed approach and cost function: *(i) resource provisioning* and *(ii) transcoding optimization*.

A. Resource Provisioning

Zhao *et al.* [4] formulated a model to minimize storage and computation costs for VoD by considering segment popularity and a weighted transcoding graph, which shows the transcoding cost for each representation from the higher representations in the representation set. Their formulation, however, did not take into account any resource constraints. To minimize the backhaul network cost, Tran *et al.* [2] modeled the collaborative joint caching and transcoding problem as an Integer Linear Program (ILP). To mitigate the proposed model's time complexity and impractical overheads, they presented an online algorithm to determine cache placement and video scheduling decisions. They extended their work to minimize the expected video retrieval delay [3]. Gao *et al.* [8] addressed resource provisioning issues for transcoding in cloud platforms to maximize financial profit. They presented a two-timescale stochastic optimization algorithm to provision resources and schedule tasks. The authors employed an open-source cloud transcoding system called Morph to implement and evaluate the proposed algorithm. Jia *et al.* [9] formulated joint optimization for caching, transcoding, and bandwidth consumption in 5G mobile networks with mobile edge computing. However, they did not consider the delay imposed by these policies. [10] tried to improve the users' QoS in terms of delivered video quality and end-to-end latency for live streaming by employing transcoding at the edge of the network. The problem is formulated as a Markov Decision Process (MDP) by considering wireless network characteristics and available resources at the edge server. Transcoding-enabled caching in the Radio Access Network (RAN) to improve the network's video capacity has been introduced in [11].

Zhang *et al.* [12] proposed a model to serve the clients' requests by video caching, transcoding, and fetching from the origin server, aiming at minimizing the average serving delay. They formulated the problem as a mixed-integer bilinear problem by considering resource constraints, *i.e.*, storage, computation, and bandwidth, at the edge server. [13] formulated transcoding-enabled caching at the edge to maximize the video provider's profit without knowing the video requests' pattern. Bilal *et al.* [14] introduced a collaborative joint caching and transcoding model using the X2 network interface [15] for sharing video data among multiple caches to minimize backhaul traffic, serving delay, and CDN cost. Lee *et al.* [16] presented a model aimed at reducing the transcoding power consumption at the edge servers by taking into account video quality factors. [17] formulated the problem of serving clients' requests by selecting a policy, including caching and transcoding at the edge and fetching from the origin server as a two-stage Stochastic Mixed-Integer Programming (SMIP) model to minimize the total energy consumption. Ref. [18] employs SDN, edge computing, and NFV capabilities to propose an SDN-based framework named S2VC. It aims to maximize QoE and QoE fairness in SVC-based HTTP adaptive streaming. The authors formulate the problem of determining the optimal bitrate and data paths for delivering the requested segments as a MILP model. Ref. [19], [20] leverage SDN and NFV capabilities to determine bitrate adaptation and flow paths for the client requests. Tashtarian *et al.* [21] propose HxL3, an architecture for low latency and QoE guarantee in Low latency Live (L3) streaming at the global Internet scale. It addresses different issues where transport protocols like TCP experience poor bandwidth and the E2E delivery path between endpoints is long. The authors provide a practical implementation of the proposed solution, and real-world experiments over the Internet show the advantage of HxL3 over its competitors in improving QoE for a given target latency. Erfanian *et al.* [22], [23] introduced OSCAR as a framework for real-time streaming. OSCAR serves clients' requests by minimizing bandwidth usage and transcoding costs. After aggregating requests at the edge servers, OSCAR transfers the highest requested bitrate from the origin server to the optimal set of Point of Presence (PoP) nodes. After that, virtual transcoders hosted at PoP nodes transcode the highest requested bitrate to those requested by clients. Finally, these bitrates are transferred to the edge servers and corresponding clients, respectively.

B. Transcoding Optimization

Jin *et al.* [24] introduced a partial transcoding approach at the edge servers. They proposed storing the full representation set for a few popular videos, keeping the highest bitrate for the rest, and preparing the other bitrates on demand by utilizing on-the-fly transcoding. They formulated the problem to minimize the total cost, including storage, transcoding, and bandwidth costs. Gao *et al.* [7] employed a partial transcoding method based on user viewing patterns in the cloud. Wang *et al.* [25] proposed an algorithm to determine edge servers for transcoding operations to improve perceived quality by clients. A distributed platform at the edge named Federated-Fog Delivery Network (F-FDN) has been introduced in [26]. Intending to minimize video streaming latency, F-FDN stores only one bitrate of non-popular videos and leverages the edge server computing power to provide requested bitrates at the edge. [27] introduced some policies for on-the-fly transcoding to improve transcoding efficiency. Li *et al.* [28] compared the impact of transcoding and bitrate-aware caching on consumers' QoE through various experiments across different bandwidth patterns, cache capacities, and popularity skewness. The authors of [29] investigated the main issues for video transcoding and formulated resource provisioning for transcoding in the cloud and caching in the network to minimize resource consumption and QoE loss. Li *et al.* [30] analyzed the transcoding performance for different types of cloud virtual machines in terms of transcoding time. Based upon the findings and by considering the cost of each virtual machine type, they also introduced a model to measure the suitability of each type of cloud virtual machine for transcoding.

In our previous work, we employed the edge computing and Network Function Virtualization (NFV) paradigms to propose a novel technique for transcoding called *L*ight-*w*eight *T*ranscoding at the *E*dge (*LwTE*) [5]. *LwTE* extracts some metadata during the encoding process and reuses it in the transcoding processes at the edge server to reduce the transcoding times and computational costs. It is worth noting that *LwTE* can be applied to most of the discussed approaches to improve their performance. Ref. [31] investigates the cost efficiency of *LwTE* in the context of live video streaming. It utilizes the proposed light-weight transcoding approach at the edge to save bandwidth in the backhaul network, which may become a bottleneck in live video streaming. In the current paper, we extend our investigations by studying *LwTE*'s performance from new aspects and relaxing some assumptions of [5]. In fact, aiming to minimize the cost and delay of VoD services, we proposed a new efficient and comprehensive BLP model by employing lightweight transcoding to serve clients' requests at the edge server. Moreover, we propose two heuristic algorithms to mitigate the time complexity of the BLP model.

III. System Model and Problem Formulation

A. General Architecture

The edge computing paradigm provides *storage* and *computation* capabilities closer to clients, resulting in a lower latency than the cloud. By employing edge resources, we propose a novel architecture to serve clients' requests by determining an *appropriate* solution, which is defined as applying a set of *policies* (*i.e.*, store, transcode and fetch) at the network edge resulting in a reduction of the overall adaptive streaming cost and delay. As depicted in Fig. 1, the proposed architecture consists of three entities named *(i)* origin server, *(ii)* edge servers, and *(iii)* HAS players. The origin server fragments a high-quality video into short-duration parts named segments. Each segment is encoded at various bitrates, yielding a bitrate ladder (representation set). We extract some encoding features, *i.e.*, metadata, during the encoding process for reuse at the edge server to avoid brute-force search in the transcoding processes and consequently reduce the transcoding times at the edge server. Note that extracting metadata during the encoding process introduces no overhead, *i.e.*, computation and delay, to the system.

A HAS player requests segments with desired bitrates, *i.e.*, resolution/quality, based on the client's parameters, *e.g.*, network condition and device properties. Each edge server provides computation, storage, and networking capabilities to support context-aware and delay-sensitive applications close to the clients. This paper employs the edge server for both caching (*i.e.*, video storage) and processing (*i.e.*, video transcoding). The proposed system for the edge server is deployed as a Virtual Network Function (VNF) and acts as a Virtual Reverse Proxy (VRP). The VRP is responsible for receiving the HAS players' requests, preparing the requested segments in desired bitrates, and finally sending them to the HAS players. The

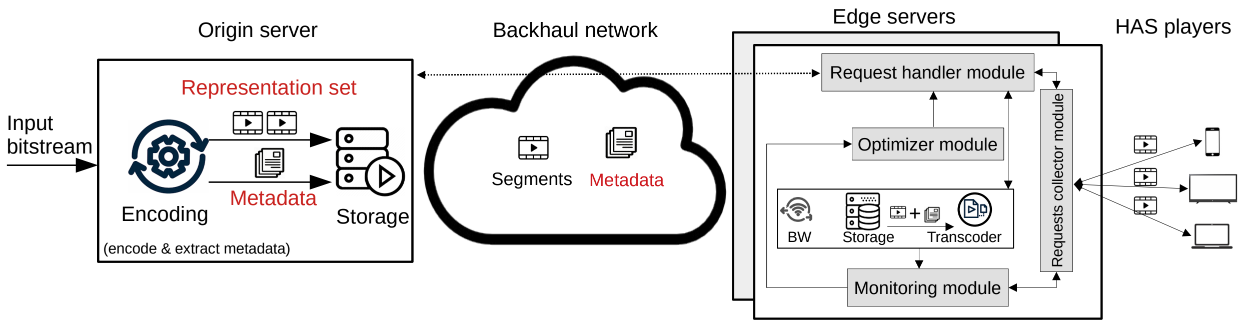


Figure 1: *CD-LwTE* architecture.

VRP serves the incoming requested segments through the following modules:

- Request Collector Module (RCM): This module is responsible for collecting the players' requests. The incoming requests are held on until the Request Handler Module prepares the requested segments/bitrates. After that, the RCM sends the requested segments/bitrates to the players. Moreover, upon receiving a request, the RCM updates the Monitoring Module.
- Request Handler Module (RHM): The RHM is responsible for serving the requested segments/bitrates using the following policies:
  - *Store*: This policy stores the *popular* segments/bitrates in the available storage at the edge server and replies to the incoming requests for them immediately from the edge server. Employing this policy introduces no delay for serving the requested segments/bitrates.
  - *Transcode*: This policy leverages the available computation resources at the edge server to prepare requested segments/bitrates by transcoding them from a higher bitrate. Therefore, the *transcode* policy introduces serving delay. Note that we need to store the metadata in case of using the *LwTE* approach for transcoding. However, the size of the metadata is very small compared to the corresponding video data.
  - *Fetch*: This policy serves the incoming requests by fetching the requested segments/bitrates from the origin/CDN server. The *fetch* policy imposes bandwidth cost and puts pressure on the backhaul network. Moreover, the *fetch* policy introduces a serving delay equal to the required time for fetching the requested segment/bitrate from the origin/CDN server, which may fluctuate due to varying traffic in the backhaul network.
- Optimizer Module (OM): This module determines an appropriate policy for each segment/bitrate by considering the parameters provided by the Monitoring Module, such as the number of incoming requests, segments'/bitrates' popularities (request probabilities), and available resources at the edge server for a time slot. The details of the OM are described in Sec. III-C.
- Monitoring Module (MM): The MM keeps the status of the resources, *i.e.*, storage, computation, and bandwidth at the edge server. Moreover, it tracks the players' behavior by storing the statistics of the incoming requests, including the segments'/bitrates' popularities. The solution determined by the OM depends on the input parameters, such as the number of incoming requests, segments'/bitrates' popularities (request probabilities), and available resources at the edge server. Thus, in case of non-trivial changes in each of the input parameters, *i.e.*, exceeding given thresholds, the MM triggers the OM to determine a new solution. For example, the OM is triggered to find a new solution when the number of requests arriving at the edge server exceeds a given threshold.

B. Metadata Extraction

Video compression standards are deploying more sophisticated tools compared to their predecessors to reduce the encoding bitrates at the same level of video quality. This improvement, however, comes at the cost of increased time complexity. For example, HEVC/H.265 compared to AVC/H.264 achieves 50% bitrate savings when encoding videos at the same video quality. In HEVC/H.265 [32], video frames are divided into fixed-size blocks named CTUs with the size of 64×64 pixels. Each CTU is then subdivided recursively into 85 Coding Units (CUs), including one 64×64 pixels CU, four 32×32 pixels CUs, sixteen 16×16 pixels CUs, and sixty four 8×8 pixels CUs. Each CU is also subdivided into Prediction Units (PUs). Each PU is then predicted from the previously encoded blocks and the prediction residuals are transformed into Transform Units (TUs) and the optimal TU size is selected after another exhaustive search. To predict each PU, in inter-coding mode, motion estimation [33] is used to find the best-matched block to reduce the residual errors. Fig. 2 shows an example of an optimal CTU partitioning into CUs after an exhaustive search.

Since the search process is very time-consuming and takes the majority of encoding time, we store the optimal decisions, including CU, PU, and TU partitioning and motion vectors as metadata, to avoid the same search process at the edge servers. The size of metadata compared to its corresponding video data is very small and leveraging it in the transcoding process at the edge results in a significant transcoding time reduction due to skipping unnecessary search processes.

C. Proposed BLP Optimization Model

The OM should determine a solution for serving requests for a time slot with a given duration of θ seconds regarding the

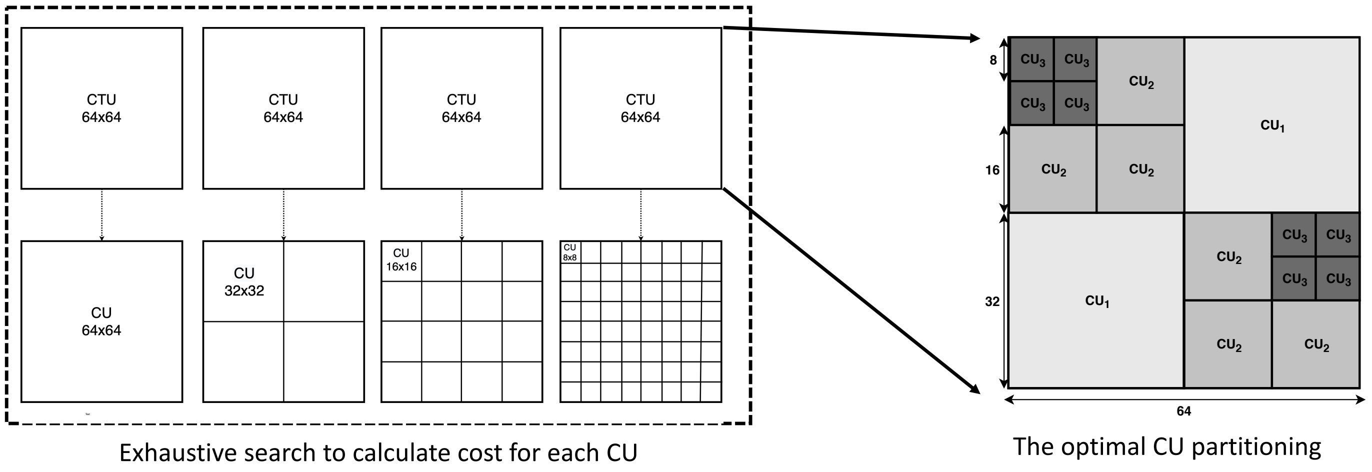


Figure 2: Each CTU is divided into CUs with different sizes and after an exhaustive search processes the optimal search decision is made.

Table I: **Notations**.

Notation	Description
	Input Parameters
$\mathcal{P}, p$	Set of possible policies p, including *store*, *fetch*, and *transcode*, to treat the requests arriving at the edge server
$\mathcal{V}$	Set of available videos
$\mathcal{S}, S_i, s_{i,j}$	Set of segments of $\mathcal{V}$, where each segment set S_i includes all segments of video i and $s_{i,j} \in S_i$ indicates the jth segment of video i
$\mathcal{K}, k$	Set of k available bitrates
$\Delta_\mathcal{S}, \Delta_\mathcal{P}, \Delta_\mathcal{B}$	Storage cost per byte per θ seconds, computation cost per CPU core per second, and bandwidth cost per byte per second, respectively
$\omega_r^{i,j}, \bar{\omega}_r^{i,j}$	Size of segment $s_{i,j}$ in bitrate r and size of corresponding metadata, respectively
$R_r^{i,j}$	Required resources (*i.e.*, CPU time in seconds) for transcoding segment $s_{i,j}$ into requested bitrate r from a higher bitrate
$F(i,j,r)$	Probability function indicating the request probability for bitrate r of segment $s_{i.j}$
$\delta_\mathcal{S}, \delta_\mathcal{P}, \delta_\mathcal{B}$	Total available storage, computation, and bandwidth resources during θ seconds, respectively
$\xi_r^{i,j}$, $\psi_r^{i,j}$	Required time for fetching bitrate r of segment $s_{i,j}$ and transcoding time for providing bitrate r of segment $s_{i,j}$ from its higher bitrate, respectively
ρ	Number of requests arriving at the server during θ seconds
	Variables
$x_{r,p}^{i,j}$	A binary variable where $x_{r,p}^{i,j} = 1$ indicates policy p is selected for serving an incoming request for bitrate r of segment $s_{i,j}$

proposed architecture and defined policies. Let $\mathcal{P}$ be the set of defined policies $\{store, transcode, fetch\}$ to be applied on the set of all segments, denoted by $\mathcal{S} = \{S_1, S_2, ..., S_v\}$, of v videos in set $\mathcal{V}$, where $S_i = \{s_{i,1}, s_{i,2}, ..., s_{i,l}\}$ consists of all segments of video $i \in \mathcal{V}$. Moreover, suppose each $s_{i,j}$ is produced in k different bitrates at the origin server. In the following, we propose a BLP model to provide a cost- and delay-aware solution which satisfies the following constraints (refer to Table I for notations):

Policy constraints: In the first group of constraints, we define two constraints to select the policy for each segment/bitrate. Let $x_{r,p}^{i,j}$ be a binary variable, where $x_{r,p}^{i,j} = 1$ indicates that policy p is selected for serving an incoming request for bitrate $r \in \mathcal{K}$ of segment $s_{i,j} \in S_i$ in video $i \in \mathcal{V}$, otherwise $x_{r,p}^{i,j} = 0$. The first constraint forces the OM to opt for only one policy for each segment/bitrate:

$$\sum_{p \in \mathcal{P}} x_{r,p}^{i,j} = 1, \qquad \forall i \in \mathcal{V}, j \in S_i, r \in \mathcal{K} \quad (1)$$

Since a higher bitrate representation is required for transcoding, the second constraint guarantees the existence of a similar segment with a higher bitrate in the case of serving the request with the *transcode* policy. Thus, by setting two policies $q = transcode$ and $p = store$, we have:

$$x_{r,q}^{i,j} \leq \sum_{b > r} x_{b,p}^{i,j}, \qquad \forall i \in \mathcal{V}, j \in S_i, b, r \in \mathcal{K} \quad (2)$$

Resource constraints: In this group of constraints, the cost of using resources (*i.e.*, processing, storage, and bandwidth) associated with the selected policies are considered. Moreover, applying policies that exceed the available amount of resources should be prevented. The cost of transcoding the given bitrate r of segment $s_{i,j}$ is obtained by multiplying the number of requests for that bitrate by the cost of transcoding. The number of requests is the total number of requests (ρ) multiplied by the request probability of bitrate r of segment $s_{i,j}$ denoted as $F(i,j,r)$, which is based on the Zipf distribution model [34].

Thus, in the case of applying the transcoding policy, the total processing cost ($\mathcal{C}_\mathcal{P}$) is defined as follows:

$$\mathcal{C}_\mathcal{P} : \sum_{i \in \mathcal{V}} \sum_{j \in S_i} \sum_{r \in \mathcal{K}} x_{r,p}^{i,j} \times \underbrace{\rho \times F(i,j,r)}_{\text{number of requests}} \times \underbrace{\Delta_\mathcal{P} \times R_r^{i,j}}_{\text{transcoding cost}},$$

where policy $p = transcode$, $\Delta_\mathcal{P}$ is the computation cost per CPU core per second, and $R_r^{i,j}$ is the required resources (*i.e.*, CPU time in seconds) for transcoding segment $s_{i,j}$ into the requested bitrate r from a higher bitrate.

The following constraint guarantees the required computation resources do not exceed the available ones (denoted by $\delta_\mathcal{P}$):

$$\sum_{i\in\mathcal{V}}\sum_{j\in S_i}\sum_{r\in\mathcal{K}} x^{i,j}_{r,p}\times\rho\times F(i,j,r)\times R^{i,j}_r \le \delta_{\mathcal{P}}, \tag{3}$$

where $p = transcode$. We similarly define the total fetch cost (bandwidth cost) in the case of applying the $fetch$ policy (*i.e.*, $p = fetch$) during θ seconds as follows:

$$\mathcal{C}_{\mathcal{B}}:\sum_{i\in\mathcal{V}}\sum_{j\in S_i}\sum_{r\in\mathcal{K}} x^{i,j}_{r,p}\times\underbrace{\rho\times F(i,j,r)}_{\text{number of requests}}\times\underbrace{\Delta_{\mathcal{B}}\times\omega^{i,j}_r}_{\text{bandwidth cost}},$$

where $\Delta_{\mathcal{B}}$ and $\omega^{i,j}_r$ are the bandwidth cost per byte per second and the size of bitrate r of segment $s_{i,j}$, respectively. The total bandwidth usage during θ seconds should be less or equal to the available one, denoted by $\delta_{\mathcal{B}}$:

$$\sum_{i\in\mathcal{V}}\sum_{j\in S_i}\sum_{r\in\mathcal{K}} x^{i,j}_{r,p}\times\rho\times F(i,j,r)\times\omega^{i,j}_r \le \delta_{\mathcal{B}}, \tag{4}$$

In our problem formulation, the storage cost, denoted by $\mathcal{C}_{\mathcal{S}}$, includes the following two items: (i) storing segments downloaded by applying the $store$ policy and (ii) storing the corresponding metadata to serve some requests using the $transcode$ policy. Thus, let $\bar{\omega}^{i,j}_r$ be the size of the corresponding metadata for transcoding segment $s_{i,j}$ from a higher bitrate to the requested bitrate r. Therefore, the maximum total storage cost is obtained as follows:

$$\mathcal{C}_{\mathcal{S}}:\sum_{i\in\mathcal{V}}\sum_{j\in S_i}\sum_{r\in\mathcal{K}}\Delta_{\mathcal{S}}\times(\underbrace{x^{i,j}_{r,p}\times\omega^{i,j}_r}_{\text{required storage for bitrate}}+\underbrace{x^{i,j}_{r,q}\times\bar{\omega}^{i,j}_r}_{\text{required storage for metadata}}),$$

where $\Delta_{\mathcal{S}}$ denotes the storage cost per byte per θ seconds, and p and q indicate applying the $store$ and $transcode$ policies, respectively. Moreover, we should limit the amount of storage consumption to the available one (denoted by $\delta_{\mathcal{S}}$):

$$\sum_{i\in\mathcal{V}}\sum_{j\in S_i}\sum_{r\in\mathcal{K}}(x^{i,j}_{r,p}\times\omega^{i,j}_r + x^{i,j}_{r,q}\times\bar{\omega}^{i,j}_r)\le\delta_{\mathcal{S}} \tag{5}$$

Delay model: As mentioned earlier, we aim at serving requests with the minimum total cost and delay. Here, we should measure the serving delay with regard to the selected policies. Thus, we formulate the total serving delay, namely $\mathcal{D}$, for two policies $p = fetch$ and $q = transcode$ as follows:

$$\mathcal{D}:\sum_{i\in\mathcal{V}}\sum_{j\in S_i}\sum_{r\in\mathcal{K}}(\underbrace{x^{i,j}_{r,p}\times\xi^{i,j}_r}_{\text{fetching delay}}+\underbrace{x^{i,j}_{r,q}\times\psi^{i,j}_r}_{\text{transcoding delay}}),$$

where $\xi^{i,j}_r$ and $\psi^{i,j}_r$ are the required time for fetching bitrate r of segment $s_{i,j}$ and transcoding of bitrate r of segment $s_{i,j}$ from a higher bitrate, respectively.

Optimization problem: Finally, the BLP model is introduced as follows:

$$\begin{aligned} &\textit{Minimize} && \frac{\alpha(\mathcal{C}_{\mathcal{P}}+\mathcal{C}_{\mathcal{B}}+\mathcal{C}_{\mathcal{S}})}{\mathcal{C}^\star}+\frac{(1-\alpha)\mathcal{D}}{\mathcal{D}^\star} \\ &s.t. && \text{Equations } (1)-(5) \\ &vars.: && x^{i,j}_{r,x}\in\{0,1\}\end{aligned}\tag{6}$$

where $0 \le \alpha \le 1$ is the weight coefficient to set desirable priorities for *(i)* the total serving cost (*i.e.*, computation, bandwidth, and storage costs) and *(ii)* the total serving delay $\mathcal{D}$. Since the total serving cost and delay have different dimensions, we normalize them by dividing them by the maximum cost and serving delay denoted by $\mathcal{C}^\star$ and $\mathcal{D}^\star$, respectively. To measure $\mathcal{C}^\star$, we run the BLP model with $\alpha = 0$ to achieve the minimum serving delay. Note that with $\alpha = 0$, the model selects appropriate policies which minimize delay and disregard serving costs. Similarly, we set $\alpha = 1$ to measure $\mathcal{D}^\star$. The obtained $\mathcal{C}^\star$ and $\mathcal{D}^\star$ are then used to run the model for different values of α. It's worth mentioning that we need to compute $\mathcal{C}^\star$ and $\mathcal{D}^\star$ for a fixed number of video sets only once.

Theorem 1: The problem of selecting an optimal policy, *i.e.*, store, transcode, and fetch for video segments/bitrates to serve clients' requests with the minimum cost and delay under edge resource constraints, *i.e.*, storage, computation, and bandwidth, is an NP-hard problem.

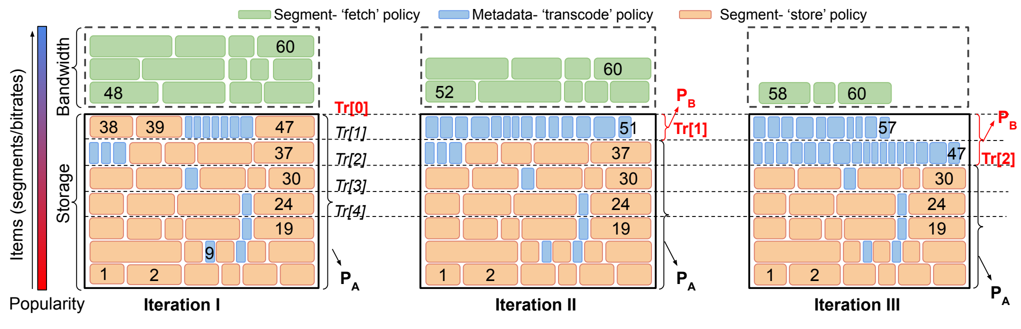
Proof: To show that a problem is NP-hard, we need to reduce an already proven NP-hard problem to the addressed problem [35], [36]. To this end, let us assume a simple case of the problem that only a limited amount of storage capacity is available at the edge. Moreover, let us consider w_i and v_i as the size of segment/bitrate i and the gained value for selecting segment/bitrate i (*i.e.*, request probability), respectively. Now, by assuming M as the maximum available storage capacity at the edge, we aim to maximize the sum of profit, *i.e.*, $\sum v_i$. Therefore, the problem can be reduced to the NP-hard 0-1 knapsack problem in polynomial time.

D. Heuristic Algorithms

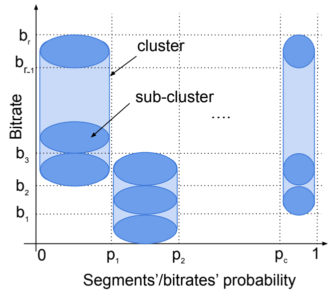
To mitigate the proposed BLP model's time complexity, we introduce the following two heuristic algorithms: (i) the *Fine-Grained* and (ii) the *Coarse-Grained* heuristic algorithms, named FGH and CGH, respectively. The main idea behind the proposed FGH algorithm is to sort segments/bitrates by their popularity and then select the right policy for each segment/bitrate separately by considering the available resources at the edge. In CGH, we diminish the search space by partitioning all segments/bitrates into a limited number of clusters.

Algorithm 1 FGH algorithm

```
Input: $Resources, Items, Tr$
1: $Items \leftarrow$ Sort($Items$)
2: $curTr \leftarrow 0$
3: $param.res \leftarrow Resources$
4: $param.startItem \leftarrow 0$
5: $rsTemp.totalObj \leftarrow \infty$
6: $checkpoints$, $result$ $\leftarrow$ CostFunc( $Items$, $rsTemp.totalObj$,
   $param$, $Tr$, $Tr[curTr]$ )
7: $rsTemp.totalObj \leftarrow result.totalObj$
8: while $rsTemp.totalObj \le result.totalObj$ && $curTr <$ len($Tr$) do
9:     $curTr \leftarrow curTr$+1
10:    $cpTemp, rsTemp$ $\leftarrow$ CostFunc( $Items$, $result.totalObj$,
       $checkpoints[curTr]$, $Tr$, $Tr[curTr]$ )
11:    if $rsTemp.totalObj \le result.totalObj$ then
12:        $result \leftarrow rsTemp$
13:    end if
14: end while
15: return $result$
```



(a)



(b)

Figure 3: (a) A simple example of the proposed FGH algorithm and (b) an illustrating example of clusters and sub-clusters.

Algorithm 2 CostFunc()

```
Input: $Items, maxObj, initParam, Tr, threshold$
1: $resources \leftarrow initParam.res$
2: $si \leftarrow initParam.startItem$
3: $result \leftarrow initParam.result$
4: $inx \leftarrow$ len($Tr$)
5: for all $i \in Items[si:]$ do
6:     $rr[0] \leftarrow \omega[i]$                                  {index 0 for $store$ policy}
7:     $rr[1] \leftarrow \rho\times F[i] \times R[i]$                 {index 1 for $transcode$ policy}
8:     $rr[2] \leftarrow \rho \times F[i] \times \omega[i]$           {index 2 index for $fetch$ policy}
9:     $cost[0] \leftarrow \Delta_{\mathcal{S}} \times rr[0]$
10:    $cost[1] \leftarrow \Delta_{\mathcal{P}} \times rr[1] + \bar{\omega}[i] \times \Delta_{\mathcal{S}}$
11:    $cost[2] \leftarrow \Delta_{\mathcal{B}} \times rr[2]$
12:    $delay[0] \leftarrow 0$
13:    $delay[1] \leftarrow \xi[i]$
14:    $delay[2] \leftarrow \psi[i]$
15:    $sp \leftarrow -1, so \leftarrow \infty$
16:    for $j = 0, 1, 2$ do
17:        $obj[j] \leftarrow \alpha \times cost[j]$ / max($cost$) + $(1-\alpha) \times delay[j]$ / max($delay$)
18:        if ($obj[j] \leq so$) and (
           ($j$=0 and $resources[0] - rr[0] > threshold$) or
           ($j$=1 and FindHS($results$,$i$) $> i$ and $resources[1] - rr[1] > 0$
           and $resources[0]$ - $\bar{\omega}[i]$ >0) or
           ($j$=2 and $resources[2] - rr[2] > 0$))   then
19:            $sp \leftarrow j$
20:            $so \leftarrow obj[j]$
21:        end if
22:    end for
23:    $result[i] \leftarrow sp$
24:    $result.totalObj \leftarrow result.totalObj + obj[sp]$
25:    $result.totalCost \leftarrow result.totalCost + cost[sp]$
26:    $result.totalDelay \leftarrow result.totalDelay + delay[sp]$
27:    $resources[sp] \leftarrow res[sp]$ - $rr[sp]$
28:    if $sp = 1$ then
29:        $resources[sp] \leftarrow resources[0]$ - $\bar{\omega}[i]$
30:    end if
31:    if $result.totalObj > maxObj$ then
32:        break
33:    end if
34:    if $resources[0] > Tr[inx]$ and $resources[0] - \omega(i+1) \leq Tr[inx]$
       then
35:        $checkpoints[inx].startItem \leftarrow i$
36:        $checkpoints[inx].result \leftarrow result$
37:        $checkpoints[inx].res \leftarrow resources$
38:        $inx \leftarrow inx - 1$
39:    end if
40: end for
41: return $checkpoints, result$
```

Assume that we have a small storage capacity compared with the given video set, and the number of arriving requests is high. In this scenario, the observation is that the proposed approaches first select the *store* policy for segments/bitrates as it results in a reasonable cost value. When the storage capacity is over, the only feasible option is the *fetch* policy since there is no storage capacity left to store bitrates and metadata for applying the *store* and *transcode* policies, respectively. In this scenario, allocating a part of the storage capacity to the *transcode* policy, *i.e.*, for storing required metadata may lead to a better solution. Therefore, we divide the storage capacity into two parts in both proposed heuristic algorithms. The first part denoted as P_A, is used for keeping *both* segments/bitrates and metadata for *store* and *transcode* policies, respectively. The second part, denoted as P_B, is *exclusively* allocated to the *transcode* policy to store the required metadata to perform transcoding tasks. Thus, determining a threshold/boundary point between P_A and P_B is an important task for the proposed heuristic algorithms. We first present the details of the FGH algorithm.

1) FGH Algorithm: Inspired by the dynamic programming approach, we introduce the FGH algorithm in two sub-algorithms. The main body of the FGH algorithm, which is depicted in Alg. 1, determines a solution by considering various thresholds between P_A and P_B values. The input parameters $Resources$, $Items$, and Tr for Alg. 1 indicate the available resources (*i.e.*, storage, computation, and bandwidth at the edge), an array of all segments/bitrates, and given l number storage thresholds between P_A and P_B, respectively. It is worth mentioning that a fine-granular step in selecting the storage threshold results in a more efficient solution; however, it introduces a higher execution time. In this study, we empirically select the number of thresholds between P_A and P_B.

In the FGH algorithm, segments/bitrates in the set $Items$ are first sorted based on their popularity. The *CostFunc()* function, described in the sub-algorithm Alg. 2, is then called to determine *(i)* the policy for each segment/bitrate in $Items$ and *(ii)* $checkpoints$ when $P_B = 0$, *i.e.*, all storage is allocated to P_A. The set $checkpoints$ has l elements that is equal to the number of thresholds between P_A and P_B (*i.e.*, Tr set in Alg. 2). Each element in the set $checkpoints$ (*i.e.*, $checkpoints[t]$ where $0 \leq t < l$) comprises the information related to the threshold t including: *(i)* the selected policies, *(ii)* available resources, *(iii)* total cost, *(iv)* delay, and (v) starting points. The set $checkpoints$ is used to skip determining policy for Items that are still in P_A in the next iterations of calling

CostFunc() (while loop in Alg. 1). In each iteration, the allocated storage to P_B is increased, based on the given values on Tr set, resulting in a lower storage for P_A. The iteration is stopped if the total objective is increased or all the given thresholds between P_A and P_B are probed.

We clarify the FGH algorithm by an example illustrated in Fig. 3 (a), where FGH should determine the solution for serving arriving requests for 60 segments/bitrates in $Items$. In Fig. 3 (a), the orange, blue, and green rectangles show the $store$, $transcode$, and $fetch$ policies, respectively.

In the first iteration, the policy set for all segments/bitrates ($Items$) are determined by calling the *CostFunc()* function. For segments/bitrates 1 to 47, *store* or *transcode* policy is selected, while for segments/bitrates 48 to 60 the fetch *policy* is selected. The information related to all thresholds are then stored in the $checkpoints$ set ($Tr[0]$ to $Tr[4]$). In the next iteration, the allocated storage for P_B is increased. Since segments/bitrates 1 to 37 are still in P_A after increasing size of P_B, their policies are remained unchanged and *CostFunc()* is called only for segments/bitrates 38 to 60. Similarly, in the next iteration, *CostFunc()* is called for segments/bitrates 31 to 60 and it is skipped for segments/bitrates 1 to 30 because they are still in P_A. Since the total objective for Iteration III is higher than its previous iteration (Iteration II), the algorithm is stopped and returns the selected policy set in Iteration II as the final policy set.

The average time complexity of the *Sort* function in Alg. 1 is $O(n \times log(n))$ where n is the number of segments/bitrates in $Items$. In the worst case, the *while* loop in Alg. 1 calls the *CostFunc()* function l times. The time complexity of *CostFunc()* is $O(n)$ in the worst case (see Alg. 2). Thus, FGH's time complexity is $O(n \times log(n) + l \times n)$, which is a function of the input parameters, *i.e.*, the number of segments/bitrates. Although FGH's time complexity is polynomial, scaling up the problem, *i.e.*, the given video set, increases its execution time. To mitigate this issue, in the following, we introduce the second heuristic algorithm CGH, with much lower time complexity in each iteration. However, CGH imposes a preprocessing step to prepare some input parameters.

2) CGH Algorithm: Based on the obtained results of the proposed FGH algorithm, we found there is a correlation between the selected policies and segments'/bitrates' popularity and size. In other words, segments with a similar popularity/size got an identical policy. Thus, the second heuristic algorithm (CGH) groups segments/bitrates into a limited number of clusters and sub-clusters, based on their popularity and bitrate, respectively. The CGH algorithm then determines the policy for each sub-cluster to serve incoming requests. The CGH algorithm consists of two sub-algorithms: In the first sub-algorithm, we perform some preprocessing operations (clustering) to prepare inputs for the main algorithm (second sub-algorithm). All segments/bitrates in the given video set are partitioned into a given number of clusters based on their popularity by the clustering sub-algorithm (Alg. 3). Segments/bitrates in each cluster are then split into a given number of sub-clusters according to their bitrates.

Algorithm 3 Clustering function

```
Input: $Items$,$m$,$btSteps$
1: $cl \leftarrow$ KmeansClustring($Items$, $m$)
2: for all $i$ in $cl$ do
3:     $c \leftarrow i.cluster$
4:     $b \leftarrow$ Findsubcluster($i$, $btSteps$)
5:     $clusters[c,b].count \leftarrow +1$
6:     $clusters[c,b].probability \leftarrow + \ i.probability$
7: end for
8: return $clusters$
```

Algorithm 4 CGH algorithm

```
Input: $Resources$,$clusters$,$Tr$
1: $clusters \leftarrow$ Sort($clusters$)
2: $curTr \leftarrow 0$
3: $param.res \leftarrow Resources$
4: $param.startCluster \leftarrow 0$
5: $rsTemp.totalObj \leftarrow \infty$
6: $checkpoints$, $result$ $\leftarrow$ CostFunc( $clusters$, $rsTemp.totalObj$,
   $param$, , $Tr$, $Tr['curTr']$ )
7: $rsTemp.totalObj \leftarrow result.totalObj$
8: while $rsTemp.totalObj \leq result.totalObj$ and $curTr < l$ do
9:     $curTr \leftarrow curTr$+1
10:    $cpTemp, rsTemp$ $\leftarrow$ CostFunc( $clusters$, $result.totalObj$,
       $checkpoints[curTr]$, $Tr$, $Tr[curTr]$ )
11:    if $rsTemp.totalObj \leq result.totalObj$ then
12:        $result \leftarrow rsTemp$
13:    end if
14: end while
15: return $result$
```

In the first line of Alg. 3, by employing the K-means algorithm [37] in the *KmeansClustering()* function, we partition segments/bitrates in $Items$ into m clusters based on their popularity (*i.e.*, request probability), in such a way that the sum of squared Euclidean distances to each cluster mean is minimized. The *for* loop (lines 2-7) puts the segments/bitrates in each cluster $i \in cl$ into some sub-clusters based on the given bitrates in $btSteps$. Fig. 3 (b) shows a simple illustration of segments/bitrates partitioned into m clusters where each cluster consists of up to r sub-clusters. Assuming the segments'/bitrates' popularity follows a long-tail access pattern, after sorting the clusters by request probability, the first clusters cover a wider request probability range. Moreover, some sub-clusters may remain empty, *i.e.*, no segments/bitrates are assigned onto them, due to request probability and bitrate distribution model. We use the "*Kmeans1d*" Python module [38] for our implementation that leverages a dynamic programming algorithm with $O(m \times n + n \times log(n))$ time complexity where n and m are the number of segments/bitrates in $Items$ and the given number of clusters, respectively. The time complexity of the sub-clustering (Alg. 3, lines 2-7) is $O(n)$. Thus the whole time complexity of the first sub-algorithm is $O(n + m \times n + n \times log(n))$.

Similar to Alg. 1, in the main CGH algorithm (Alg. 4), we find a near-optimal solution and storage threshold. However, Alg. 4 determines the policy for each sub-cluster in $clusters$ instead of each segment/bitrate. Note that the input parameter $clusters$ is provided by Alg. 3. In line 1, we sort the input set $clusters$ based on the popularity attribute in $O(x \times log(x))$ time, where x denotes the number of sub-clusters and equals $x = m \times r$. Then the proposed algorithm (Alg. 4) calls the *CostFunc()* function l times (the number of threshold values in Tr) in the worst case (lines 8-14). The *CostFunc()* function determines the solution for each sub-cluster; thus, its time complexity is $O(x)$ in the worst case, in which all sub-clusters have at least one segment/bitrate. So the time complexity of Alg. 4 is agnostic to the video set size and equals $O(x \times log(x) + l \times x)$, which is much lower than FGH's time complexity since l and x are relatively small numbers. Note that we set $clusters$ as an input for *CostFunc()* (Alg. 2) instead of $Items$ and determine the suitable policy for each sub-cluster. It is worth mentioning that Alg. 3 is called only one time in the system initialization process and is kept updated by the MM. It means that, in case of insertion/evacuation of a segment/bitrate, we need to update the $count$ and $probability$ attributes in the corresponding sub-cluster. When the OM is triggered, we need to call only Alg. 4 for determining an appropriate solution.

IV. Performance Evaluation

A. Evaluation Setup

To evaluate the proposed approach, we transcode videos at 30 fps to the full set of bitrates $\{r1, r2, ..., r12\}$ according to [39] using the HEVC/H.265 open-source encoder x265 version 3.4, where $r1$ and $r12$ have the highest and lowest bitrate/quality, respectively. The "medium" encoder preset is used as the default; however, we investigate the performance of *CD-LwTE* under different *presets*. To consider the impact of the *video content* on the transcoding time and, consequently, the proposed algorithm, we perform the transcoding for various sequences with different *complexities* (*i.e.*, "easy to encode" and "hard to encode") at different *segment lengths* (*i.e.*, four and six seconds). We also cover various *video lengths*, ranging from 2 to 120 minutes. Our experiments assume that the number of source videos and the video popularity will remain unchanged, and clients can request any bitrate of a segment from the bitrate ladder. For simplicity, we also assume that the access probability of each video is known in advance and follows a Zipf-like distribution [34]. The probability of requesting each segment/bitrate is calculated based on [5].

The storage, computation, and bandwidth costs are 0.024\$ per GB per month, 0.029\$ per CPU per hour, and 0.12\$ per GB, respectively [40]. In our experiments, we measure the results for a time slot with a one-hour duration ($\theta = 3600$ seconds). Transcoding operations are performed on Amazon EC2 instances. We use $c5.2xlarge$ as the default instance type [40], but repeat our experiments on various Amazon EC2 instances to investigate the performance of *CD-LwTE* with different *computation power profiles*, *i.e.*, CPU cores and RAM. In our simulation, we set the bandwidth between the origin/CDN server and the edge server to 1 Gbps. The storage capacity at the edge server is set to 1 TB. The edge server is equipped with four CPU cores and a $c5.2xlarge$ Amazon EC2 instance. Moreover, we consider 50 equal size steps, *i.e.*, Tr in Alg. 1 and Alg. 3, from 0% to 50% of the storage, for allocating storage capacity to the $transcode$ policy. The number of clusters (m) and number of sub-clusters in each cluster (r) in Alg. 3 are set to 200 and 12, respectively. Furthermore, we set the default value of coefficient parameter α in Eq. 6 to 0.5.

We evaluate the performance of *CD-LwTE* in six scenarios. In scenario I, we investigate the performance of *CD-LwTE* for different transcoding aspects. In scenario II, the performance gaps between the proposed BLP model and the proposed heuristic algorithms are measured in terms of cost, average serving delay, and execution time. In scenario III, we evaluate the performance of the proposed heuristic algorithms for different numbers of requests and videos. Scenario IV examines the performance of the proposed heuristic algorithms for various storage profiles. In scenario V, we explore the performance of the proposed heuristic algorithms for various weight coefficient values. In the last scenario, we compare the performance of the heuristic algorithms with state-of-the-art approaches in terms of total cost and average serving delay.

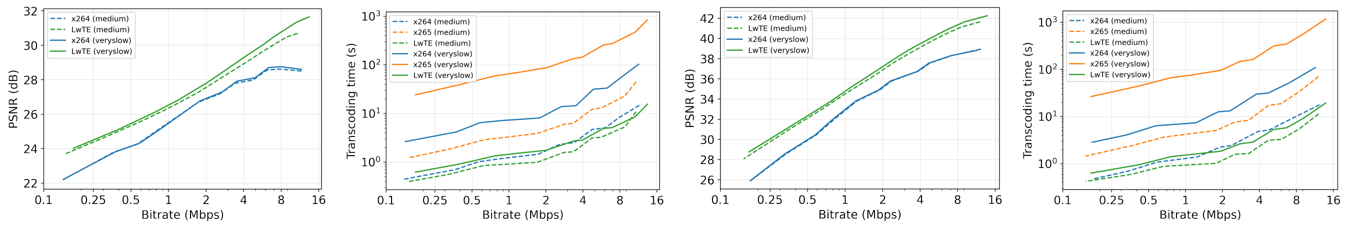
B. Scenario I

Advanced Video Coding (AVC) [41], or H.264, is widely used in the industry due to its low transcoding time. HEVC/H.265 [42], the successor of AVC, shows higher compression efficiency but higher transcoding time compared to the AVC standard. The transcoding time is more crucial at the edge as the processing resources are limited. However, by leveraging metadata, *CD-LwTE* results in a significant reduction of transcoding time.

We compare the compression efficiency of AVC/H.264 using the x264 open-source encoder to HEVC/H.265 using the x265 open-source encoder employing metadata (LwTE) and without use of metadata (x265). Fig. 4 (a) and Fig. 4 (c) show compression efficiency of the above-mentioned methods for both *medium* and *veryslow* presets for the $ParkRunning3$ and $FoodMarket4$ video sequences, respectively. Note that use of metadata impacts only transcoding time, not compression efficiency. Therefore, the compression efficiency of x265 and LwTE (without and with metadata, respectively) remains the same; consequently, only one of them (LwTE) is plotted. LwTE (x265) yields bitrate savings of 48% and 52% to maintain the same PSNR as compared to x264, for *ParkRunning3* and *FoodMarket4*, respectively.

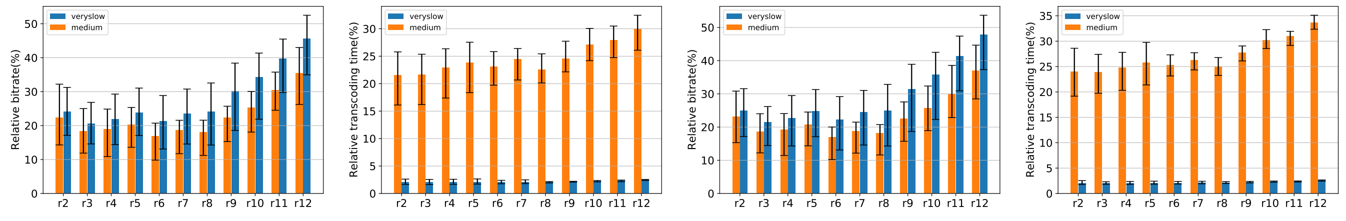
In Fig. 4 (b) and Fig. 4 (d), the transcoding times of x264, x265, and LwTE for both *medium* and *veryslow* presets are compared for the *ParkRunning3* and *FoodMarket4* sequences, respectively. It is seen that employing the metadata significantly reduces the transcoding times of x265, even to lower times than for x264. For example, for *veryslow* and *medium* presets, the transcoding time of LwTE is, on average, 87% and 48% less than x264, respectively.

To compare the size of the metadata to its corresponding video segment representation, we plot the bitrate of the metadata relative to its corresponding segment bitrate for the entire bitrate ladder, *i.e.*, $\{r2, ..., r12\}$, for 4-sec. and 6-sec. segments in Fig. 5 (a) and Fig. 5 (c), respectively. The relative bitrate of each representation was obtained by averaging the relative bitrates of five video sequences, namely, $ParkRunning3$, $FoodMarket4$, $Maples$, $Basketball$, and $Bunny$. The maximum and minimum relative bitrates were also added to the plots. Roughly 70% to 80% bitrate (storage) saving is obtained by replacing the segment/bitrate video data



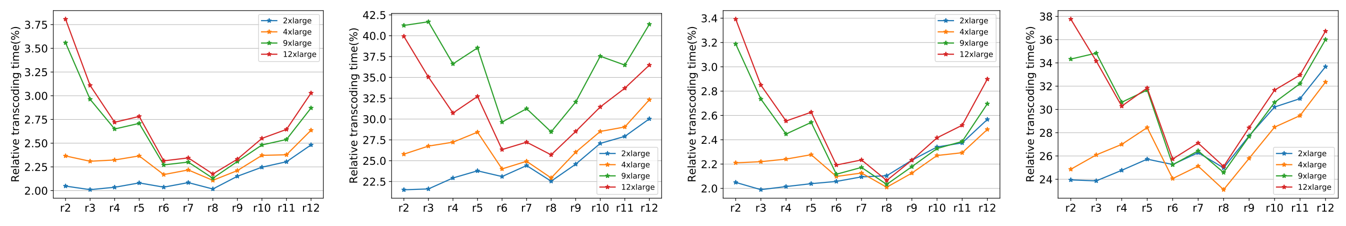
(a) (b) (c) (d)

Figure 4: Compression efficiency of x264 and x265 (without metadata) and x265 with metadata employed (LwTE) for (a) *ParkRunning3* and (c) *FoodMarket4* sequences and *medium* and *veryslow* presets. Note that x265 and LwTE have the same compression efficiency. Transcoding times of x264, x265, and LwTE for (b) *ParkRunning3* and (d) *FoodMarket4* sequences and *medium* and *veryslow* presets.



(a) (b) (c) (d)

Figure 5: Average bitrates of metadata relative to its corresponding representations for (a) 4-sec. and (c) 6-sec. segments. Average transcoding times of “x265 with metadata ” relative to “x265 without metadata” for (b) 4-sec. and (d) 6-sec. segments.



(a) (b) (c) (d)

Figure 6: Average relative transcoding times over five sequences for (a) 4-sec. segments and the veryslow preset, (b) 4-sec. segments and the medium preset, (c) 6-sec. segments and the veryslow preset, (d) 6-sec. segments and the medium preset, on different AWS EC2 instances.

with its corresponding metadata in particular for high- and mid-bitrate representations, which consume much more storage space than the low-bitrate representations, where 50% to 70% bitrate saving is achieved. The transcoding times of “x265 with metadata” relative to “x265 without metadata” for 4-sec. and 6-sec. segments are shown in Fig. 5 (b) and Fig. 5 (d), respectively. The relative transcoding times were also obtained by averaging over the above-mentioned sequences, and the maximum and minimum relative times were added to the plots. The use of metadata results in more than 97% transcoding time saving for the *veryslow* preset and approximately 70% for the *medium* preset.

We also evaluate the impact of running encoders on different AWS instances with different numbers of CPU cores and RAM profiles, namely, *c5.2xlarge*, *c5.4xlarge*, *c5.9xlarge*, and *c5.12xlarge*. Fig. 6 shows the relative transcoding times for different presets and instances averaged over the five above-mentioned sequences. Transcoding on the *c5.2xlarge* instance yields the highest transcoding time saving.

C. Scenario II

This scenario compares the proposed approaches (FGH, CGH, and BLP model) based on the cost and average serving delay for different α values to illustrate the performance differences and the impact of weight coefficient values (Fig. 7(a) and (b)). To this, we set the available storage at the edge $\delta_{\mathcal{S}}$= 500 GB, computation resource $\delta_{\mathcal{P}} = 8$ CPU cores, $|V| = 50$, $\rho = 5\,k$, and $\alpha = \{0, 0.2, 0.5, 0.8, 1\}$. CPLEX [43] is employed as a solver to run the proposed BLP model in Python. However, CPLEX could determine the solution only for limited video sets in a reasonable time due to the high time complexity. Note that the serving delay is introduced by applying the $fetch$ and $transcode$ policies when serving segment/bitrate requests. The serving delay for the segments/bitrates with the $store$ policy is zero since the incoming requests are served from the edge cache server. Thus, the average serving delay is defined as the average of the introduced delays by various policies (*i.e.*, including $store$ policy with zero delays) for serving the incoming requests to the edge. To minimize the delay, the proposed approaches select the $store$ policy as much as the storage capacity constraint allows. Moreover, the cost metric shows the total cost of serving arriving requests at the edge server during a time slot, including storage, computation, and bandwidth costs ($\mathcal{C}$ in Eq. 6). Selecting an appropriate policy for minimizing the total cost mainly depends on the number of incoming requests (*i.e.*, $\rho \times F(i,j,r)$). Obviously, for the popular segments/bitrates, the $store$ policy is a better option than other policies. However, for the unpopular ones, the proposed approaches must select one of the $transcode$ or $fetch$ policies based on resource constraints and the number of incoming requests to each segment/bitrate. We have comprehensively investigated the problem of determining the boundary point between popular and unpopular segments/bitrates in [5].

As illustrated in Fig. 7(a) and Fig. 7(b), the BLP has higher sensitivity to α values than the FGH and CGH algorithms due to its fine granular nature for determining the solution. By setting $\alpha = 0$, the proposed approaches try to minimize the serving delay; thus, they select the *store* policy for the segments/bitrates as much as possible (*i.e.*, considering the storage capacity). That is, they result in the highest cost and the minimum average serving delay at $\alpha = 0$ (see Fig. 7(a) and (b)). By increasing to $\alpha = 0.2$, there is almost no change in the proposed schemes' cost and average serving delay. For $\alpha = 0.5$, decreasing the total cost has more priority for our schemes; therefore, they reduce the total cost, resulting in a rise in the measured average serving delay. Although FGH and CGH outperform the BLP model in terms of cost at this point, their average serving delay values are much higher than that of the BLP model. That is, the BLP model determines the optimal solution for all segments/bitrates in the video set while the FGH and CGH algorithms specify a near-optimal solution for each segment/bitrate and each sub-cluster, respectively; this leads to a sub-optimal solution overall but quite close to the BLP model for the whole video set. For $\alpha = 1$, BLP shows the minimum cost while its average serving delay is higher than that of the FGH algorithm.

To measure the execution time for the proposed approaches, we run them with various video sets $|V|$= $(5, 10, 20, 50, 10, 500, 1000, 5000)$. As depicted in Fig. 7(c), the BLP model suffers from high time complexity, and its execution time increases exponentially with increasing the video set size. On the other hand, the FGH algorithm results in almost a linear growth in the execution time when increasing the size of the video set. CGH outperforms the other approaches significantly in terms of execution time, showing very small growth with increasing video set size. Note that the CPLEX solver could not determine the solution for video sets with more than 50 sequences in the given time frame, *i.e.*, in less than a time slot duration (one hour). However, we run the BLP model for the video set with 100 sequences, and as it is depicted in Fig. 7(c), the execution time lasts for around 23 hours. Thus, we do not measure BLP's performance with bigger video sets; therefore, there are no values for the BLP model for these video set sizes.

D. Scenario III

As mentioned earlier, some input parameters, like the number of arriving requests, are time-varying and unknown in advance. The MM tracks the state of input parameters like available resources, incoming requests rate, and segments'/bitrates' popularities and triggers the OM to find a new solution in case of non-trivial changes in the input parameters. In this scenario, we investigate the proposed heuristic algorithms' performance for input parameters, including video set size and the numbers of arriving requests in terms of storage allocation, average serving delay, cost per time slot, and percentage of segments/bitrates and requests served by each policy. In this direction, we take into account various video set sizes (100, 500, 1000, and 5000 sequences) and numbers of arriving requests ($\rho = \{5\,k, 10\,k, 50\,k, 100\,k, 150\,k\}$) per time slot (one hour).

It is obvious from Fig. 8 that the proposed algorithms show quite similar performance on these metrics.

Fig. 8(a) and Fig. 8(b) illustrate the average serving delay and serving cost for the proposed algorithms, respectively, where both rise sharply by increasing the video set. For $|V| = 100$, the FGH and CGH algorithms utilize the storage capacity at 61% to 71% and 62% to 68% for various ρ values, respectively, while they serve only 33% to 40% and 33% to 42% of segments/bitrates by the $store$ policy (Fig. 8(c)). In the case of $|V| \geq 500$, both proposed algorithms fully utilize the available storage capacity; thus, they have to serve more segments/bitrates and consequently more incoming requests by the $transcode$ and $fetch$ policies with higher serving delay and cost (see Fig. 8 (c)–(d)). This means that by taking into account the segments'/bitrates' popularity, applying the $store$ policy on all segments/bitrates is not a cost-efficient solution. Moreover, as there is sufficient storage capacity when increasing the ρ value, the proposed approaches store more segments/bitrates, which results in a significant reduction in the average serving delay while the total cost increases slightly ($|V| = 100$ in Fig. 8 (a)–(b)). On the other hand, in the case of $|V| > 100$, the proposed approaches consume all the available storage to minimize the cost function (Eq. 6). However, the cost rises sharply when increasing the number of arriving requests (ρ) while the average serving delay remains almost steady. As depicted in Fig. 8 (c)–(d), by storing a small portion of segments/bitrates, mainly for $|V| > 100$, the proposed algorithms can serve the majority of incoming requests that stem from the long-tail access pattern.

It is obvious that selecting an appropriate policy for each segment/bitrate is dependent on various parameters (*i.e.*, cost and delay in the current study), environment constraints (*i.e.*, available storage, computation, and bandwidth resources), and access frequency/probability (see Section III-C). For example, for $|V| = 100$ the proposed algorithms do not fully utilize the storage capacity. Likewise, the rate of serving by the *transcode* policy that relies on different parameters, does not show desirable performance. For example, the *store* policy is useful for popular segments/bitrates (*i.e.*, high access rate) while the *transcode* and *fetch* policies are suitable for unpopular ones. In this scenario, the proposed algorithms mainly select the $transcode$ policy to serve a limited number of incoming requests for $|V| > 100$ while the transcoding rate decreases with increasing the number of incoming requests.

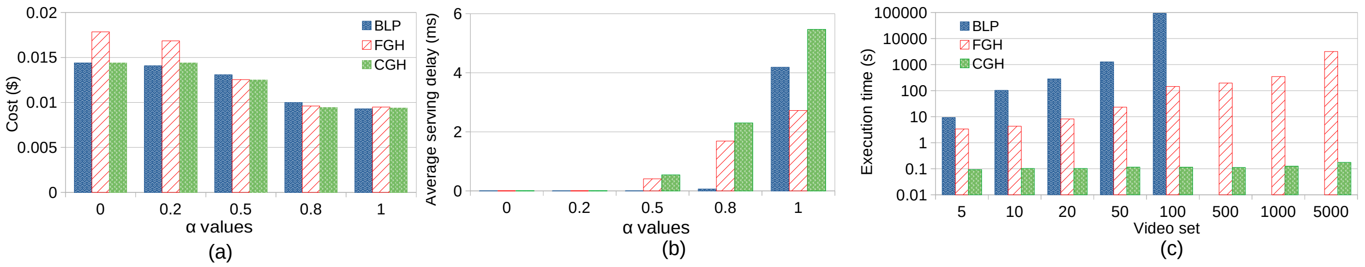


Figure 7: Performance of the proposed *CD-LwTE* approaches for different α values in terms of (a) cost and (b) average serving delay, and (c) execution time for various video sets.

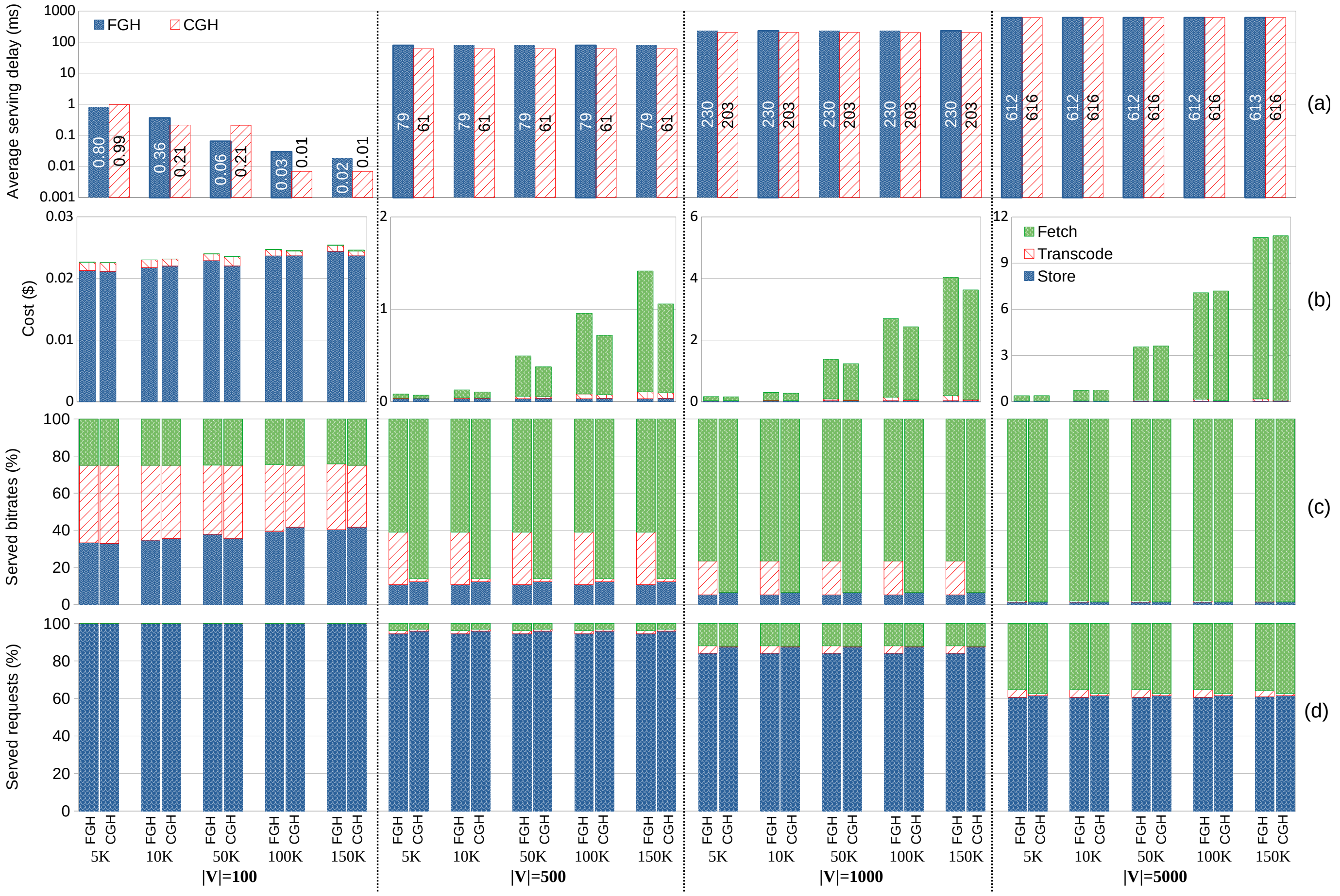


Figure 8: Performance of the proposed heuristic algorithms for different video sets in terms of (a) storage allocation, (b) average serving delay, (c) cost, (d) percentage of segments/bitrates served by each policy, and (e) percentage of requests served by each policy.

E. Scenario IV

In this scenario, we study the performance of the proposed heuristic algorithms for various storage profiles, *i.e.*, storage capacity, in terms of resource utilization, average serving delay, cost, and percentage of segments/bitrates and requests served by each policy. We set the available storage at the edge $\delta_{\mathcal{S}} = \{500\,GB, 1\,TB, 1.5\,TB, 2\,TB, 3\,TB\}$. Moreover, the computation resource $\delta_{\mathcal{P}}$ is set to 8 CPU cores. We measure the *CD-LwTE* performance for $|V| = 1000$ and various ρ values. The proposed algorithms consume all the available storage capacity for all values of ρ and $\delta_{\mathcal{S}}$. The FGH algorithm consumes more computation resources and results in higher values in terms of average serving delay (Table II). Fig. 9 (a) shows the cost. It is obvious that by increasing the storage capacity, the values obtained for the average serving delay and cost reduce significantly since more requests can be served using the $store$ policy (Fig. 9 (b)). In fact, Fig. 9 (b) shows that in case of limited storage capacity at the edge, employing the $transcoding$ policy results in cost and delay reduction. However, in case of sufficient storage capacity at the edge and lower ρ values *e.g.*, $\delta_{\mathcal{S}} = 3\,TB$ and $\rho \leq 10\,k$, the efficient approach is to serve part of the arriving requests by the $transcode$ policy.

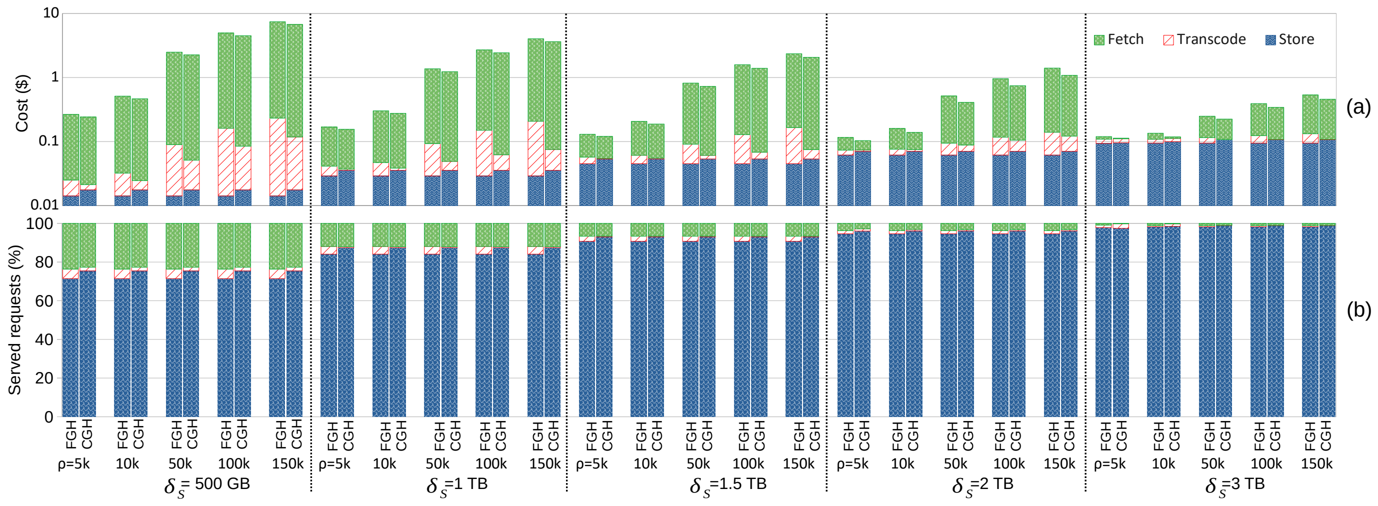


Figure 9: Performance of the proposed *CD-LwTE* approaches for different storage sizes in terms of (a) cost, (b) percentage of the requests served by each policy.

F. Scenario V

In the last scenario, we compare the performance of the proposed heuristic algorithms in terms of cost, cache hit ratio, and bandwidth utilization in the backhaul network with the following state-of-the-art approaches:

- APCP [2]: This approach serves the incoming requests through the $store$, $transcode$, and $fetch$ policies w.r.t. minimizing the serving delay in the cost function.
- CoCache: This approach is a simplified version of the proposed algorithm in [3] that does not consider the $transcode$ policy. In fact, CoCache tries to minimize the serving delay by selecting the $store$ policy on the segments/bitrates as much as possible, then applies the $fetch$ policy on the rest.
- PartialCache [4]: This approach minimizes the cost by applying the $store$ and $transcode$ policies on segments/bitrates until the available storage and computation resources are fully utilized. After that, it serves the remaining segments/bitrates with the $fetch$ policy.
- PartialCache (x264): The state-of-the-art approaches listed above use the x265 encoder in a conventional manner, *i.e.*, without metadata for transcoding. Aiming at comparing the performance of the proposed heuristic algorithms with state-of-the-art approaches employing the x264 encoder as a fast and widely use encoder in the industry, we measure PartialCache's performance when using the x264 encoder. For fair comparisons, we encode the considered videos with the same quality as x265 does, by using the x264 encoder. As expected and illustrated in Fig. 4, the videos encoded using x264 come up with around 50% higher bitrate and transcoding time at the same quality of x265.

Table II: Performance of the proposed *CD-LwTE* approaches for different storage size in terms of average serving delay (ms).

δ_S	Alg.	ρ				
		5k	10k	50k	100k	150k
500 GB	FGH	429	429	429	429	429
	CGH	386	386	0.21	386	386
1 TB	FGH	230	230	230	230	230
	CGH	203	203	203	203	203
1.5 TB	FGH	132	132	132	132	132
	CGH	111	111	111	111	111
2 TB	FGH	76	76	76	76	76
	CGH	58	60	60	60	60
3 TB	FGH	25	24	24	24	24
	CGH	22	17	18	18	18

Note that all the considered state-of-the-art approaches determine the policy by taking into account the segments/bitrates list sorted by popularity, ρ, and the available resources at the edge server as the input parameters. We define the cache hit ratio as the fraction of requests that can be served either from the local storage at the edge server or by transcoding from a higher bitrate. As depicted in Fig. 10 (a), the CGH algorithm results in the best cost for $\rho \leq 10\,k$, and reduces the cost up to 75% compared with state-of-the-art approaches for $\rho = 10\,k$. The FGH algorithm achieves the best cost for $\rho > 10\,k$; however, the CGH algorithm has the second best cost and surpasses the other approaches for $\rho > 10\,k$. On the other hand, the FGH algorithm outperforms all other approaches in terms of average serving delay for $\rho \leq 10\,k$ (*i.e.*, reduces the average serving delay up to 48%) , while the CGH algorithm has the best average serving delay for $\rho > 10\,k$ (see Fig. 10 (b)). Moreover, the CGH and FGH algorithms show the best performance in terms of the cache hit ratio for $\rho \leq 10\,k$ and $\rho > 10\,k$, respectively (see Fig. 10 (c)). Among the studied state-of-the-art approaches, *PartialCache* shows better performance in terms of cost (Fig. 10 (a)). However, it results in the worst performance in terms of cost, average serving delay, and cache hit ratio when employing the x264 encoder for performing the transcoding operations (*PartialCache (x264)*). In terms of average serving delay, the *APCP* and *CoCache* approaches have the best cost values, following the proposed *CD-LwTE* algorithms (Fig. 10 (b)). However, *PartialCache* shows better performance on cache hit ratio than the other state-of-the-art approaches (Fig. 10 (c)).

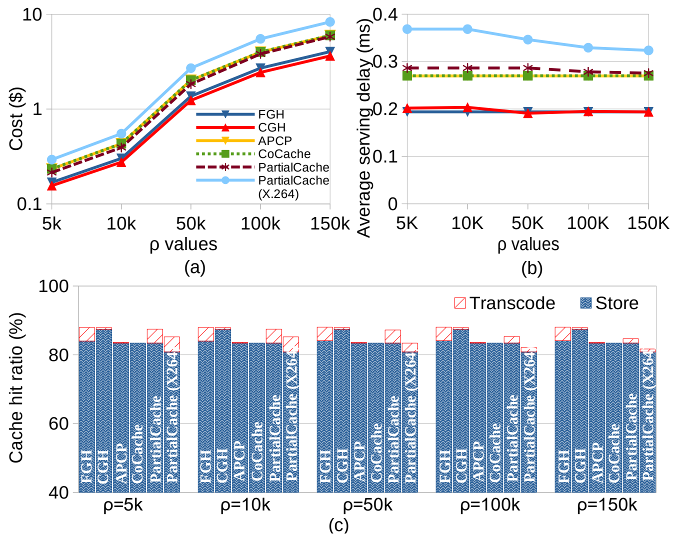


Figure 10: Performance of the proposed *CD-LwTE* approaches compared with state-of-the-art approaches in terms of (a) cost, (b) average serving delay, and (c) cache hit ratio, for various ρ values.

V. Conclusion and Future Work

In this study, we extend the investigations of our previous work [5] by proposing *CD-LwTE*, a *C*ost- and *D*elay-aware *L*ight-*w*eight *T*ranscoding approach at the *E*dge, in the context of HTTP Adaptive Streaming. *CD-LwTE* stores the optimal search results of the encoding process as metadata for each bitrate of a video segment and employs it at the edge servers to avoid brute force search in the transcoding processes, aiming at reducing the transcoding cost. We formulate the problem of selecting an optimal policy for serving segment/bitrate requests at the edge server for VoD services, including (*i*) storing at the edge server, (*ii*) transcoding from a higher bitrate at the edge server, and (*iii*) fetching from the origin or a CDN server, as a Binary Linear Programming (BLP) model with the cost function of minimizing the cost and delay. Indeed, *CD-LwTE* stores the popular segments/bitrates at the edge and serves the unpopular ones by transcoding, employing the metadata, or fetching from the origin/CDN server. Moreover, we prove the proposed BLP model is an NP-hard problem and propose two heuristic algorithms to mitigate the high time complexity of the BLP model. We investigate the performance of *CD-LwTE* in comprehensive scenarios with various video contents, encoders, encoding settings, and available resources at the edge. The experimental results show that our approach (*i*) reduces the transcoding time by up to 97%, (*ii*) decreases the streaming cost, including storage, computation, and bandwidth costs, by up to 75%, and (*iii*) reduces delay by up to 48% compared to state-of-the-art approaches. The future work direction is to devise an online approach by leveraging the reinforcement learning method.

Acknowledgment

This research has been supported in part by the *Christian Doppler Laboratory ATHENA* (https://athena.itec.aau.at/).

References

[1] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "Analyzing the Video Popularity Characteristics of Large-scale User Generated Content Systems," *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1357–1370, 2009.

[2] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," in *2017 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. IEEE, 2017, pp. 165–172.

[3] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 1965–1978, 2018.

[4] H. Zhao, Q. Zheng, W. Zhang, B. Du, and H. Li, "A Segment-based Storage and Transcoding Trade-off Strategy for Multi-version VoD Systems in the Cloud," *IEEE Transactions on Multimedia*, vol. 19, no. 1, pp. 149–159, 2016.

[5] A. Erfanian, H. Amirpour, F. Tashtarian, C. Timmerer, and H. Hellwagner, "LwTE: Light-weight Transcoding at the Edge," *IEEE Access*, 2021.

[6] F. Jokhio, A. Ashraf, S. Lafond, and J. Lilius, "A Computation and Storage Trade-off Strategy for Cost-efficient Video Transcoding in the Cloud," in *39th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2013, pp. 365–372.

[7] G. Gao, W. Zhang, Y. Wen, Z. Wang, and W. Zhu, "Towards Cost-efficient Video Transcoding in Media Cloud: Insights Learned from User Viewing Patterns," *IEEE Transactions on Multimedia*, vol. 17, no. 8, pp. 1286–1296, 2015.

[8] G. Gao, H. Hu, Y. Wen, and C. Westphal, "Resource Provisioning and Profit Maximization for Transcoding in Clouds: A Two-timescale Approach," *IEEE Transactions on Multimedia*, vol. 19, no. 4, pp. 836–848, 2016.

[9] Q. Jia, R. Xie, H. Lu, W. Zheng, and H. Luo, "Joint Optimization Scheme for Caching, Transcoding and Bandwidth in 5G Networks with Mobile Edge Computing," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. IEEE, 2019, pp. 999–1004.

[10] Z. Zhang, R. Wang, F. R. Yu, F. Fu, and Q. Yan, "Qos aware transcoding for live streaming in edge-clouds aided hetnets: An enhanced actor-critic approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 295–11 308, 2019.

[11] H. A. Pedersen and S. Dey, "Enhancing mobile video capacity and quality using rate adaptation, RAN caching and processing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 996–1010, 2015.

[12] T. Zhang and S. Mao, "Joint Video Caching and Processing for Multi-Bitrate Videos in Ultra-Dense HetNets," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 1230–1243, 2020.

[13] Y. Hao, L. Hu, Y. Qian, and M. Chen, "Profit maximization for video caching and processing in edge cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 7, pp. 1632–1641, 2019.

[14] K. Bilal, E. Baccour, A. Erbad, A. Mohamed, and M. Guizani, "Collaborative joint caching and transcoding in mobile edge networks," *Journal of Network and Computer Applications*, vol. 136, pp. 86–99, 2019.

[15] 3GPP, "LTE; Evolved Universal Terrestrial Radio Access Network (E-UTRAN); X2 general aspects and principles (3GPP TS 36.420 version 15.2.0 Release 15) ," *White Paper*, January 2020.

[16] D. Lee, J. Lee, and M. Song, "Video quality adaptation for limiting transcoding energy consumption in video servers," *IEEE Access*, vol. 7, pp. 126 253–126 264, 2019.

[17] L. Li, D. Shi, R. Hou, R. Chen, B. Lin, and M. Pan, "Energy-efficient proactive caching for adaptive video streaming via data-driven optimization," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5549–5561, 2020.

[18] F. Tashtarian, A. Erfanian, and A. Varasteh, "S2VC: An SDN-based framework for maximizing QoE in SVC-based HTTP adaptive streaming," *Computer Networks*, vol. 146, pp. 33–46, 2018.

[19] Erfanian, Alireza and Tashtarian, Farzad and Yaghmaee, Mohammad Hossein, "On maximizing QoE in AVC-based HTTP adaptive streaming: An SDN approach," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[20] Erfanian, Alireza and Tashtarian, Farzad and Timmerer, Christian and Hellwagner, Hermann, "QoCoVi: QoE-and cost-aware adaptive video streaming for the Internet of Vehicles," *Computer Communications*, vol. 190, pp. 1–9, 2022.

[21] Tashtarian, Farzad and Bentaleb, Abdelhak and Erfanian, Alireza and Hellwagner, Hermann and Timmerer, Christian and Zimmermann, Roger, "HxL3: Optimized Delivery Architecture for HTTP Low-Latency Live Streaming," *IEEE Transactions on Multimedia*, 2022.

[22] A. Erfanian, F. Tashtarian, R. Farahani, C. Timmerer, and H. Hellwagner, "On Optimizing Resource Utilization in AVC-based Real-time Video Streaming," in *6th IEEE International Conference on Network Softwarization (NetSoft)*, Ghent, Belgium, June 2020.

[23] A. Erfanian, F. Tashtarian, A. Zabrovskiy, C. Timmerer, and H. Hellwagner, "OSCAR: On Optimizing Resource Utilization in Live Video Streaming," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 552–569, March 2021.

[24] Y. Jin, Y. Wen, and C. Westphal, "Optimal Transcoding and Caching for Adaptive Streaming in Media Cloud: An Analytical Approach," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 1914–1925, 2015.

[25] Z. Wang, L. Sun, C. Wu, W. Zhu, Q. Zhuang, and S. Yang, "A Joint Online Transcoding and Delivery Approach for Dynamic Adaptive Streaming," *IEEE Transactions on Multimedia*, vol. 17, no. 6, pp. 867–879, 2015.

[26] V. Veillon, C. Denninnart, and M. A. Salehi, "F-FDN: Federation of fog computing systems for low latency video streaming," in *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2019, pp. 1–9.

[27] D. K. Krishnappa, M. Zink, and R. K. Sitaraman, "Optimizing the video transcoding workflow in content delivery networks," in *Proceedings of the 6th ACM Multimedia Systems Conference*, 2015, pp. 37–48.

[28] W. Li, S. M. Oteafy, and H. S. Hassanein, "Performance Comparison of Transcoding and Bitrate-aware Caching in Adaptive Video Streaming," in *IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[29] G. Gao and Y. Wen, "Video transcoding for adaptive bitrate streaming over edge-cloud continuum," *Digital Communications and Networks*, 2020.

[30] X. Li, M. A. Salehi, Y. Joshi, M. K. Darwich, B. Landreneau, and M. Bayoumi, "Performance analysis and modeling of video transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 910–922, 2018.

[31] A. Erfanian, H. Amirpour, F. Tashtarian, C. Timmerer, and H. Hellwagner, "Lwte-live: Light-weight transcoding at the edge for live streaming," in *Proceedings of the Workshop on Design, Deployment, and Evaluation of Network-Assisted Video Streaming*, ser. VisNEXT'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 22–28. [Online]. Available: https://doi.org/10.1145/3488662.3493829

[32] E. Çetinkaya, H. Amirpour, M. Ghanbari, and C. Timmerer, "CTU depth decision algorithms for HEVC: A survey," *Signal Processing: Image Communication*, vol. 99, p. 116442, Nov. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0923596521002113

[33] H. Amirpour, M. Ghanbari, A. Pinheiro, and M. Pereira, "Motion estimation with chessboard pattern prediction strategy," *Multimedia Tools and Applications*, vol. 78, no. 15, pp. 21 785–21 804, Aug. 2019. [Online]. Available: https://doi.org/10.1007/s11042-019-7432-8

[34] L. Cherkasova and M. Gupta, "Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change," *IEEE/ACM Transactions on networking*, vol. 12, no. 5, pp. 781–794, 2004.

[35] Martello, Silvano and Toth, Paolo, "Algorithms for knapsack problems," *North-Holland Mathematics Studies*, vol. 132, pp. 213–257, 1987.

[36] J. Erickson, "Np-hard problems," *Algor. Course Mat*, vol. 21, pp. 1–18, 2009.

[37] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[38] https://pypi.org/project/kmeans1d/, accessed: 2021-11-19.

[39] "HLS Authoring Specification for Apple Devices," https://developer.apple.com/documentation/http_live_streaming/hls_authoring_specification_for_apple_devices.

[40] https://calculator.aws/, accessed: 2021-10-30.

[41] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

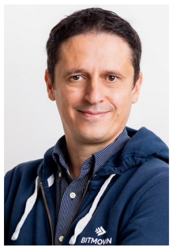
[42] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[43] IBM ILOG, *V12.8: User's Manual for CPLEX*, International Business Machines Corporation, 2017.

Alireza Erfanian is a Ph.D. student in the ATHENA projects at the Institute of Information Technology (ITEC), Alpen-Adria-Universität Klagenfurt (AAU). He received the B.Sc. and M.Sc. degrees in computer engineering in 2009 and 2017, respectively. He has been working in the computer networks field for over twelve years. His research interests are multimedia communication and adaptation, software-defined networking, network function virtualization, 5G networks, and optimization. Further information at https://sites.google.com/view/alireza-erfanian.

Hadi Amirpour is a postdoc research fellow at Christian Doppler (CD) Laboratory ATHENA based at the University of Klagenfurt. He received his Ph.D. in computer science from the University of Klagenfurt in 2022. He received two B.Sc. degrees in Electrical and Biomedical Engineering and an M.Sc. degree in Electrical Engineering from the K. N. Toosi University of Technology. He was involved in the project EmergIMG, a Portuguese consortium on emerging imaging technologies, funded by the Portuguese funding agency and H2020. His research interests include: video streaming, image and video compression, quality of experience, emerging 3D imaging technology and medical image analysis. Further information at https://hadiamirpour.github.io

Farzad Tashtarian (M'15) is a post-doctoral researcher in the ATHENA project at the Institute of Information Technology (ITEC), Alpen-Adria-Universität Klagenfurt (AAU). Before joining the team, he was an assistant professor at the Azad University of Mashhad, Iran. He received his Ph.D. from the Ferdowsi University of Mashhad in Computer Engineering. He is a member of the Technical Program Committee of several international conferences. His current research areas of interest are end-to-end latency and QoE in video streaming, video networking, software- defined networking, network function virtualization, mathematical modeling, and distributed optimization. Further information at https://tashtarian.net/.



Christian Timmerer (M'08-SM'16) is a full professor of computer science at the Institute of Information Technology (ITEC) and is the director of the Christian Doppler (CD) Laboratory ATHENA (https://athena.itec.aau.at/). His research interests include immersive multimedia communication, streaming, adaptation, and quality of experience where he co-authored seven patents and more than 300 articles. He was the general chair of WIAMIS 2008, QoMEX 2013, MMSys 2016, and PV 2018 and has participated in several EC-funded projects, notably DANAE, ENTHRONE, P2P-Next, ALICANTE, SocialSensor, COST IC1003 QUALINET, ICoSOLE, and SPIRIT. He also participated in ISO/MPEG work for several years, notably in the area of MPEG-21, MPEG-M, MPEG-V, and MPEG-DASH where he also served as standard editor. In 2013 he cofounded Bitmovin (http://www.bitmovin.com/) to provide professional services around MPEG-DASH where he holds the position of the Chief Innovation Officer (CIO) — Head of Research and Standardization. Further information at http://timmerer.com.

Hermann Hellwagner is a full professor of computer science at Alpen-Adria-Universität Klagenfurt (AAU) where he leads the research group Multimedia Communication (MMC) in the Institute of Information Technology (ITEC). Earlier, he held positions as an associate professor at the University of Technology in Munich (TUM) and as a senior researcher at Siemens Corporate Research in Munich. His current research areas are distributed multimedia systems, multimedia communication and adaptation, QoS/QoE, information-centric networking, and communication in multi-UAV networks. He has published widely on parallel computer architecture, parallel programming, and multimedia communication and adaptation. He is a senior member of the IEEE and a member of the ACM. He was a member of the Scientific Board of the Austrian Science Fund (FWF) (2005-2016) and an FWF Vice President (2013–2016). Currently, he is a member of the CD Senate of Christian Doppler Forschungsgesellschaft (CDG). Further information at https://www.itec.aau.at/~hellwagn/.