Edge Assisted DASH Video Caching Mechanism for Multi-access Edge Computing

Shashwat Kumar, Doddala Sai Vineeth, and Antony Franklin A.
Computer Science and Engineering
Indian Institute of Technology Hyderabad
Telangana, India
[cs15resch11011, cs14btech11011, antony.franklin]@iith.ac.in

***Abstract*—Proliferation in mobile devices and the increase in video data consumption on these devices has led to an unprecedented surge of data usage in mobile networks. It is both challenging and expensive for network operators to scale up the network capacity and tackle this ever increasing data demand. Cellular network operators require alternative solutions, like in-network-caching, to solve this problem. Popular streaming services like YouTube use Dynamic Adaptive Streaming over HTTP (DASH) for video streaming where videos are divided into several small segments, and multiple bit-rate versions of each segment are stored in the server. Using store and forward caching method, in the network, may not help as the video segments cached in one session might not be usable for other users. This problem of unusability emerges as different users request different bit-rates of the same video segment. Also, it is not efficient to cache all versions of the video segments at the edge of the network, due to limited storage at the edge. In this paper, we propose a Multi-access Edge Computing (MEC) based video caching mechanism, where only the highest available bit-rate video is cached and by using the processing power available at the MEC it is transcoded to the requested lower bit-rate version. We develop a test-bed to evaluate the performance of the proposed caching mechanism in real time. Through various experimental results, we demonstrate that the proposed method reduces the backhaul traffic load and video load time and increases the cache hit-rate as compared to traditional store and forward caching mechanism.**

***Index Terms*—edge caching, MEC, video caching, caching testbed**

I. Introduction

According to Cisco [1], global IP traffic will reach 278 EB per month and video traffic will have a share of 82% in all consumer traffic by 2021. This excessive data usage will cause back-haul congestion in the cellular network and thereby driving the network operators to provide measures that tackle network congestion. Extending the network capacity is one of the plausible solutions to mitigate the congestion, however, due to the exponential growth in data demand and high capital expenditure (CAPEX), this approach is not sustainable. Edge caching is a prominent solution for serving popular content from the network edge, especially video content. If the video is served from the network edge, the back-haul network congestion due to video traffic can be avoided. Edge caching can be enabled in the cellular network with Multi-access Edge Computing (MEC) [2]. MEC is an architecture that provides storage and computation at the network edge (i.e., base station) for deployment of applications and services.

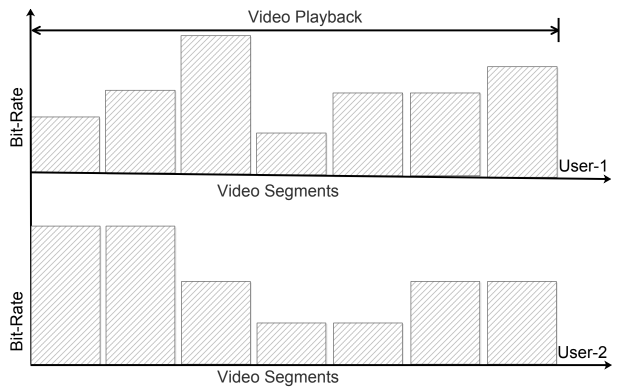


Fig. 1: Bit-rates of different segments during a video playback for two different users in a cellular network.

Content providers use Adaptive Bit Rate (ABR) video streaming for video services. Dynamic Adaptive Streaming over HTTP (DASH) has been developed as a standard for ABR with the aim to improve video Quality of Experience (QoE). DASH splits each video into multiple segments of equal playtime (i.e., 2sec.) and multiple bit-rate versions of the same video are stored in the content server. During video playback, DASH tries to improve the users' Quality of Experience (QoE) by selecting the bit-rate of the next video segment, which needs to be downloaded, in such a way that it minimizes the probability of re-buffering and stalling. This quality selection is primarily influenced by users' preferences, device capabilities, and network conditions. Availability of multiple bit-rates makes video caching for DASH quite challenging. As shown in Figure 1, during the playback of a video, two different users may request different bit-rates of the same video segment. Hence, cached video segments from one session are unusable to serve other users watching the same video. A simple caching mechanism such as store and forward is not efficient for caching the DASH videos. To overcome the limitations on DASH video caching, we exploit the transcoding technique that allows the conversion of a higher bit-rate video to a lower bit-rate video. MEC cache manager always caches the highest available bit-rate of the video for the reusability of the cached video segments for different video bit-rate requests. When a user requests for a lower bit-rate version of a video,

video segments are transcoded to the requested lower bit-rate version from the higher bit-rate video to serve the user. In this work, we use the cache consolidation and cache splitting solution as proposed in [3]. To realize the cache consolidation, we introduce a central cache controller which helps in sharing the cache information among the MEC servers. Our major contributions in this work include:

- Development of a testbed for the evaluation of video caching mechanism in a MEC architecture and performance evaluation of the proposed caching mechanism on the developed testbed.
- Evaluation of the feasibility of the real-time transcoding with $ffmpeg$ [4]. To evaluate the proposed caching method, we created a data set of 200 videos and validated it for conformance with ISO/IEC 23009-1 MPEG-DASH.

Rest of the paper is organized as follows. Section II covers the significant work in this area. In Section III, we provide the details of the developed testbed architecture and discuss the proposed video cache algorithm. Section V highlights the details of the experimental test-bed. In Section VI, we discuss the results and finally conclude the paper in Section VII.

II. Related Work

In [5], Lei et al. implemented a prototype for edge using Virtual Network Function (VNF) and realized the edge caching through the service chaining using Open Air Interface (OAI) and Content Centric Networking (CCNx). In [6], authors used proactive caching off popular content at the edge during off-peak hours to alleviate backhaul congestion. They further used the social structure of the network and D2D communication to disseminate the content. Authors in [7], proposed an edge caching scheme in the mobile network based on content-centric-networking. They utilized the backhaul link between the Base Stations (BSs) for collaborative caching. In [8], caching and processing for multi-bitrate video streaming is proposed. However, they do not consider the collaborative scheme of multiple caching servers. The heuristic solution in [9] requires the knowledge of the content popularity, which may be hard to estimate accurately in practice. Most of the works do not solve the problem of DASH video caching at the edge in the test-bed environment and mostly rely on the simulation results which might differ when applied in the real world.

III. System Architecture

Figure 2 shows the proposed architecture where MEC servers are deployed alongside the eNBs in cellular Radio Access Network (RAN), providing computation and storage capabilities to enable caching at the edge of the network. MEC server can also be deployed in Evolved Packet Core (EPC), but that may increase the latency, so we choose MEC deployment at eNB. MEC servers collaborate to share their computing and storage resources. The proposed cache architecture has the following components;

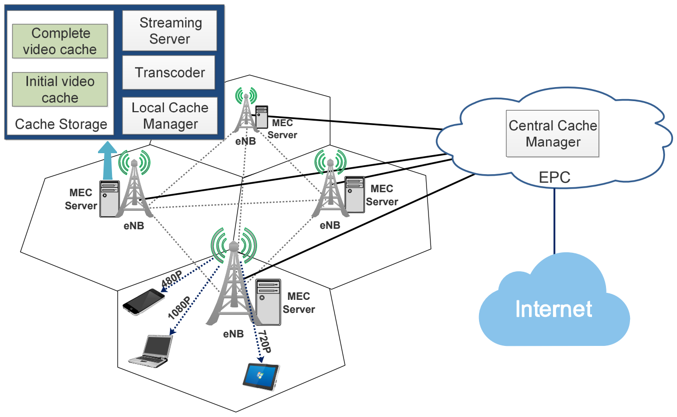


Fig. 2: System architecture of edge caching using Multi-access Edge Computing (MEC).

A. *Cache Storage*

Cache storage caches the video segments for future use. Cache storage is bifurcated into complete-video-cache and initial-video-cache. Complete-video-cache stores all the segments of a video and initial-video-cache stores only initial segments, sufficient to start the playback, of a video. Cache splitting is further discussed in Section IV.

B. *Central Cache Manager*

Central cache manager keeps track of all the videos cached in the MEC network with a hash map of video ID and their location. Central cache manager ensures the cache consolidation among the MEC servers and avoids replication of the same video in the cache network. The central cache manager provides the video location information to local cache managers and receives cache updates from local cache manager whenever there is a change in the complete video cache of that MEC.

C. *Local Cache Manager*

Local cache manager resides on the MEC server at the network edge. On receiving a video request, local cache manager search for the video in the local caches on MEC server. If the video is not available locally, then local cache manager sends the request to central cache manager for the video location and forward the received location to the streaming server. When a video is added to or get replaced in the local cache, local cache manager propagates this information to the central cache manager with update messages. These update messages carry either add_video message to make an entry for a new video or del_video message to remove the entry of a replaced video.

D. *Streaming Server*

The streaming server on each MEC streams the requested videos to the users. The streaming server fetches the video segments from the location (local cache, other MEC servers, or main content server) provided by the local cache manager. If video segments are fetched from the main content server, streaming server caches the video segments in the complete video cache. When streaming server fetches video segments

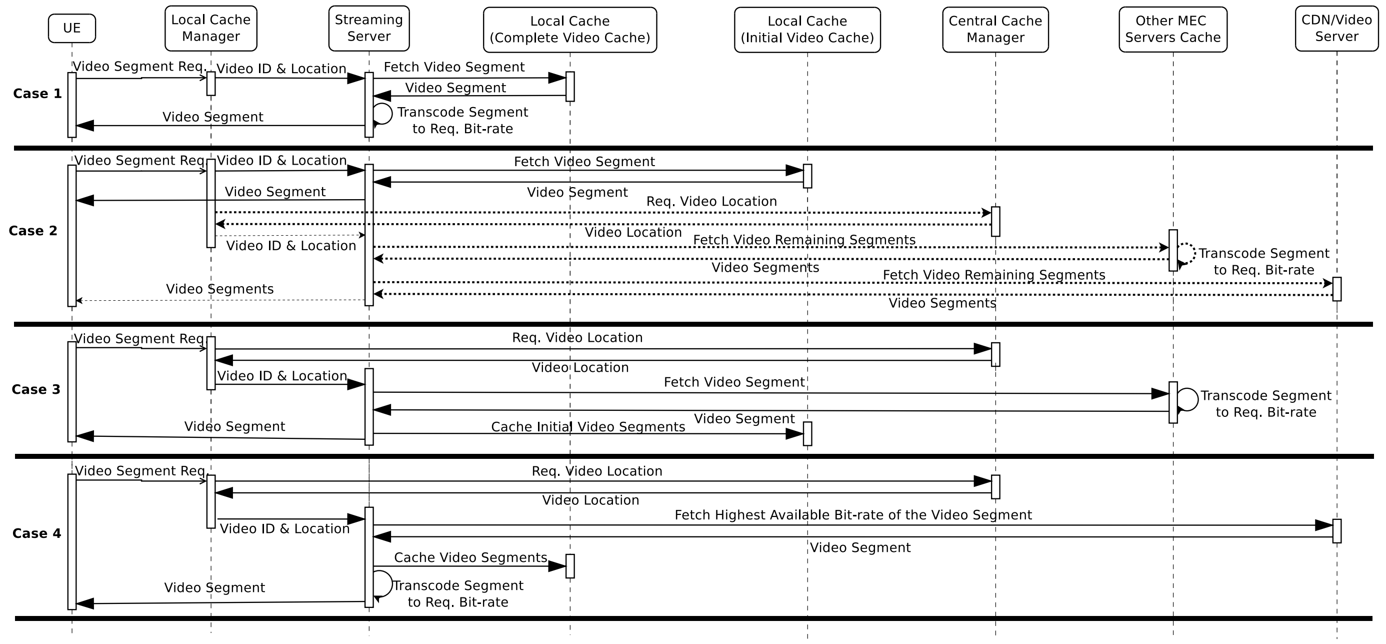


Fig. 3: Sequence diagram representing the different scenarios to fulfill the user requests.

from the cache of other MEC server, it caches initial segments of the video in the initial video cache on local MEC server.

E. Transcoder

Transcoder converts a video segment from high bit-rate to a requested lower bit-rate version. Video segments are transmitted in $.m4s$ format to play on a DASH client, but .m4s video segment cannot be transcoded to the lower bit rate as it is. For transcoding, the video segment is converted to $.mp4$ format by adding the $dashinit$ header of the same bit-rate to the .m4s data. Now, $.mp4$ file is transcoded to the required lower bit-rate version using $FFmpeg$ [4]. After transcoding, video segments are converted back to .m4s format using $MP4box$ [10] so they can be played on a DASH client.

Figure 3 shows the sequence diagram of the operations followed to deliver a video segment through the proposed architecture in different scenarios. On receiving a request, one of the four cases is followed based on the availability of the video content.

Case 1: When the complete video is available in the local cache. Local cache manager forwards the video request to the streaming server and streaming server fetches the video segments from the local cache and serves the user after transcoding it to the requested bit-rate.

Case 2: When only initial video segments are available in the local MEC server, the streaming server serves the initial segments from the local cache and fetches the remaining segments from either other MEC server or main content server. If video segments are fetched from the main content server then streaming server cache the complete video locally.

Case 3: When the video is not available on local MEC server, local cache manager sends a query to the central cache manager which provides the location of the MEC server with cached video. While serving the user from other MEC server, the streaming server also caches the initial segments of the video in the local cache.

Case 4: When the requested video segment is not available in the cache network, central cache manager returns the address of the main content server upon receiving the query from the local cache manager. In this case, the streaming server caches all the video segments in the local complete video cache while serving the user.

IV. Proposed Solutions for Video Caching

In the proposed solution, we try to use both caching and processing capabilities at the MEC servers to satisfy the user requests for different video bit-rates. With sufficient processing power to transcode the video from higher bit-rate version to a lower bit-rate version, there is no need to cache lower bit-rate videos. In this work, we propose that only the highest available bit-rate h of the video segments s is cached at the edge of the network. When a user requests for a lower bit-rate $q : q<h$ of the video segment v_q^s, it is served after transcoding from the higher bit-rate version v_h^s. Algorithm 1 shows the working of the proposed solution, where C_j^F and C_j^I represent the set of all videos cached in complete and initial video cache at MEC server j. In step 2 and 3, video segment v^s is available in local cache (C_j^F or C_j^I) and the user is served from the local cache. In step 4, if the segment is cached on other MEC server C_k^F then it is fetched from there after transcoding to v_q^s. As proposed in [3], we use cache consolidation and cache splitting in this work. A brief description of cache consolidation and cache splitting is as follow,

Algorithm 1 Caching Algorithm

```
1: For each video segment request $v_q^s$ arriving at MEC server on eNB $j$, proceed.
2: if $v_h^s \in C_j^F$ then transcode the video segment ($v_h^s \rightarrow v_q^s$) and serve the user from $C_j^F$.
3: else if $v_q^s \in C_j^I$ then serve the video segments from $C_j^I$.
4: else if $v_h^s \in C_{j,j \neq k}^F$ then Fetch the video segment after transcoding ($v_h^s \rightarrow v_q^s$) at server k. If $v_q^s$ is initial video segment then cache it in $C_j$ ($C_j^I \rightarrow C_j^I + v_q^s$).
5: else
6:     fetch the video segment ($v^s$) from origin server in highest available bit-rate ($v_h$) and cache it on $C_j^F$ ($C_j^F \rightarrow C_j^F + v_h^s$), transcode the video segment ($v_h^s \rightarrow v_q^s$) and serve the user.
7: end if
```

Cache Consolidation

As shown in Figure 2, edge caches reside on MEC servers at the eNbs which results in a distributed video caches. MEC servers can collaborate to consolidate the distributed cache storage. In cache consolidation, the same video does not replicate over different MEC servers and when a request arrives for a video, MEC servers can share the content to fulfill the request. With cache consolidation, MEC servers can cache more videos collectively which improves the hit ratio and external traffic load.

Cache splitting

During video playback, initial load time, the time between the user request and starting of playback, depends on how fast user receives the initial video segments. Caching initial video segments are sufficient to reduce the initial load time. However, if all cache storage caches only the initial video segments then, for each hit, rest of the video segments need to be downloaded from the content server over the Internet. This will result in high external traffic load on back-haul. To balance the external traffic load and the access delay, cache storage can be logically bifurcated into two parts. One part of the cache storage is used to cache complete videos and another part to cache only initial segments of the videos. When some videos are cached on some other MEC server, initial segments of the video are cached locally in initial video cache to reduce the initial load time.

V. Experimental Setup

For the experiment, we deployed one main video server, four MEC servers, and 20 clients. Each MEC server serves five clients. The main video server is deployed on a workstation with two Xeon CPU E5-2690 processor with a clock speed of 2.60 GHz, 40 cores, and 1TB hard disk drive (HDD) storage. Each MEC server runs on a virtual machine with ten cores each with 2.60 GHz clock speed. Local cache manager and a local streaming server run on each MEC server and each MEC server has the cache storage that we vary for the experiments. Each MEC servers is connected to the network with 1Gbps links. For creating a real-life scenario, we add a delay of 10ms between MEC servers and 20ms between the main server and MEC servers. Netem and Wondershaper tools are used to simulate a cellular environment on the links between MEC server and client.

As discussed earlier, the main components in the architecture are central cache manager, local cache manager, and local streaming server. The central cache manager is a multi-threaded python program that waits for a connection request from the local cache manager using the socket programming. Central cache manager uses a hash table to store the video ids and their location. When local cache manager caches a video in complete video cache, it sends an add_video command to central cache manager with video ID to add an entry of the video in the central hash table. When local cache manager replaces a video in the complete video cache, it sends a del_video command to central cache manager to delete the video entry from the hash table. If a video is not available in the local cache, local cache manager sends a $query$ command to find the location of the video. The location returned by central cache manager might be one of the MEC servers if the video is stored in the cache network or main video server.

Local cache manager uses python flask module to handle requests from the client browser. If the video is not cached locally, local cache manager connects to the central cache manager to find the video location. When the central cache manager returns the video location, local cache manager returns it to the client browser in an HTML page which also contains the address of the videos' mpd file. The HTML page follows the standard DASH forum format so any browser, which supports DASH streaming, can play the video. The streaming server is implemented in Node.js and Express web application framework is used for streaming application. Local cache manager listens to users video requests on a pre-specified socket.

TABLE I: Video resolutions and respective encoding bit-rates.

Resolution	Bit-rate
240p (320x240)	0.4Mbps
360p (640X360)	0.8Mbps
480p (854X480)	1.5Mbps
720p (1280X720)	3Mbps
1080p (1920X1080)	5Mbps

Popular video streaming service providers such as YouTube use HTTPS to transfer the videos using end-to-end encryption, so it is not possible to cache the content from that service provider. To test a video caching mechanism a broad set of videos are required, so we created a data set of 200 videos in DASH format using H.264 encoding. Table I shows the bit-rates used for encoding; these encoding rates are comparative to the representation and resolution of YouTube [11].

Our data set has the videos in five resolutions and three segment sizes (2 Secs, 4 Secs, and 6 Secs). So our data set contains 15 copies of a video in different resolutions and segment sizes. We generate Media Presentation Description (MPD) file for all the videos using template-based segment

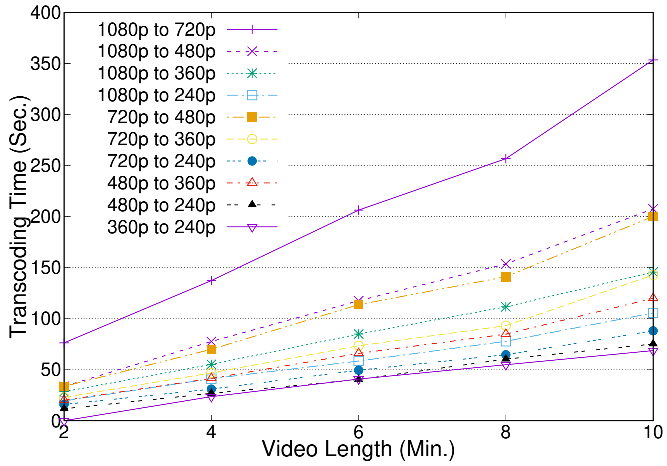


Fig. 4: Time taken to transcode videos of different playtime from a higher bit-rate input video to a lower bit-rate video.

representation in MPD files. All generated MPD files have been validated with the online DASH validator of "dashif.org" for conformance with ISO/IEC 23009-1 MPEG-DASH.

VI. Results and Analysis

In this section, we evaluate the performance of the proposed scheme. An MEC server is deployed at each eNB, as shown in Figure 2, to provide the caching and processing resources. We conducted the experiments with the video library of 200 videos and playtime of each video is 3-8 minutes. We evaluated the proposed caching method using Independent Reference Mode (IRM), where content i is requested according to an independent Poisson process with a rate λp_i. $\lambda = 7minute$ is used to control the request rate and p_i refers to content popularity. The user requests follow the Zipfs' popularity distribution [12], which gives the probability of an incoming request for i^{th} popular video as,

$$p_i = \frac{i^{-\alpha}}{\sum_{j=0}^{N} j^{-\alpha}} \tag{1}$$

where α is Zipf parameter which is set to 0.8.

To use the transcoding mechanism along with caching, we should know the scalability of the transcoding. If a transcoder can transcode multiple videos to a lower bit-rate version in real-time than we can cache only highest available bit-rate version and convert it to lower bit-rate whenever required. As some resources can be allocated for the transcoding on the MEC servers, to evaluate the number of possible parallel transcoding, we use two machines one with 8-Cores and other with 4-Cores where each core runs at 2.4Ghz clock speed. GPU is not available on either machine to assist in encoding. When encoding speed goes below $1X$ (means transcoding taking more time than the playback time of the video), we consider it as no more parallel encoding is possible on the machine.

Results in Table II show that when we transcode the video from one higher bit-rate to lower bit-rate (each row shows the input video bit-rate and each column shows the output

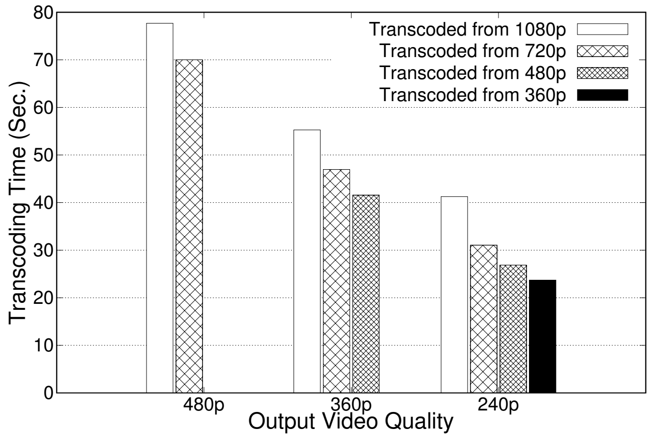


Fig. 5: Time taken to transcode a video to a lower bit-rate video from different higher bit-rate videos.

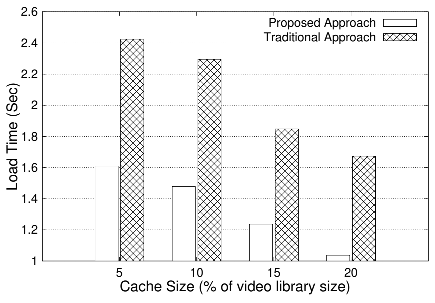
TABLE II: Number of real-time parallel transcodings from higher to lower resolution on 4 and 8 core machines.

	With 8-Core Machine				With 4-Core Machine			
	720p	480p	360p	240p	720p	480p	360p	240p
1080p	3	6	10	14	2	3	4	6
720p	-	7	12	18	-	4	6	9
480p	-	-	14	22	-	-	8	11

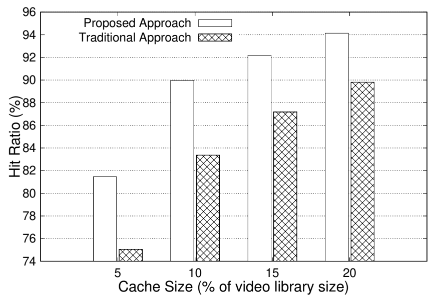
video bit-rate). The number of possible parallel conversions depend on the input and output video bit-rates. When we convert a video from $1080P$ to $240P$, the machine supports 14 parallel conversions but this value decreases when output bit-rate increases ($360P$ - 10, $480P$ - 6, and $720P$ - 3). When we transcode from different higher bit-rate inputs videos to a lower bit-rate video, conversion from lower bit-rate input video supports more parallel transcodings. As from the Table II column three $480P$ bit-rate input video provides the maximum number of transcoding followed by $720P$ and $1080P$. The reason behind this behavior is that transcoder needs to process more information when it converts from $1080P$ to $360P$ compared to $480P$ to $360P$ conversion. As we double the number of cores for transcoding, the number of parallel transcoding increase.

Figure 4 shows the time to transcode a higher bit-rate video to lower bit-rate video. As average video length on YouTube is 160s [12], so we choose a ten-minute video for transcoding time calculation and take initial two, four, six, eight minutes and full video to calculate the transcoding time. To calculate the transcoding time, we transcoded entire videos at a time or else only one segment is transcoded at a time in real-time. Results show that transcoding time increase with the video duration and output video bit-rate. Figure 5 shows that when we transcode a video to a predefined bit-rate from different bit-rate input videos, transcoder takes less time to transcode the lower bit-rate input video compared to the higher bit-rate video. Transcoding time may differ for different videos based on the scene complexities.

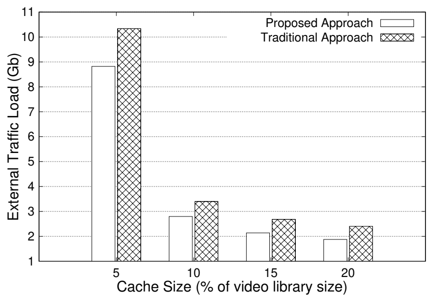
We compare the proposed method with the traditional



(a) Load Time



(b) Hit Ratio



(c) External Traffic Load

Fig. 6: Change in load time, hit ratio, and external traffic load with an increase in cache size.

approach. The traditional caching scheme is store and forward approach where the video segments are cached in the requested quality and same video may be cached on different MECs causing the data replication. Because store and forward method is predominantly used for caching in real-world so we compare the proposed caching scheme only with the traditional approach. We examine the two methods based on the following parameters;

- Load Time - Load time is the time taken between the users' request for video and when playback starts.
- Hit ratio - fraction of requests fulfilled from the MEC cache network.
- External traffic load - the amount of data fetched from CDN/content server to fulfill the user requests.

Figure 6(a) shows that the proposed approach significantly decreases the load time. In the proposed method, only the highest available bit-rate of the video is cached at the edge and it is transcoded to the requested bit-rate for every request before serving the user. By caching only the highest available bit-rate, more number of videos are cached on the MEC servers because only one bit-rate variant of each video is cached. Even though transcoding adds some marginal delay, on average it reduces the load time significantly by 33%-38%.

Figure 6(b) shows that the proposed approach provides better hit-ratio compared to traditional approach. Proposed approach increase in cache hit-ratio by 5%-8%. As we cache only the highest available bit-rate and perform the transcoding for requests of other bit-rates, more videos are cached. In the traditional approach, as multiple bit-rates of the same video might be available in the cache, it affects the hit-ratio as less distinct videos can be cached.

Figure 6(c) shows the percentage of data downloaded from the content server to fulfill the user requests. For 20% of the cache size on each MEC server, proposed approach decrease in the external traffic up to 48%.

VII. Conclusion

In this work, we proposed that the cache should store only the highest available bit-rate video and use the transcoding to serve the lower bit-rate video requests. To evaluate the performance of the proposed caching scheme in real time, we developed a test-bed for DASH video caching and prepared a dataset of 200 videos (encoded in H.264) in DASH format. First, we evaluated the transcoding with $ffmpeg$ and results show that the number of possible parallel transcoding depends on the input and output video bit-rates and without any hardware acceleration multiple video streams can be transcoded in real-time. The experimental results for the proposed caching approach show a significant improvement in the hit-ratio and up-to 22% decrease in the external traffic load. Proposed caching scheme also improves the users Quality Of Experience (QoE) by reducing the video load time up-to 38% with a cache size of 20% of the total video library.

VIII. Acknowledgement

This work has been supported by the Department of Telecommunications, Ministry of Communications, India as part of the Indigenous 5G Test Bed project.

References

[1] Cisco, "Cisco Visual Networking Index, Global mobile data traffic forecast update, 2016-2021," *white paper*, February 2017.

[2] ETSI, "Mobile-Edge Computing - Introductory Technical White Paper," *white paper*, September 2014.

[3] S. Kumar and A. F. A., "Consolidated caching with cache splitting and trans-rating in mobile edge computing networks," in *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec 2017.

[4] "FFmpeg," 2018. [Online]. Available: https://www.ffmpeg.org/

[5] L. Lei, X. Xiong, L. Hou, and K. Zheng, "Collaborative edge caching through service function chaining: Architecture and challenges," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 94–102, JUNE 2018.

[6] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, August 2014.

[7] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5G systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, February 2014.

[8] B. Shen, S.-J. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Transactions on Multimedia*, vol. 6, no. 2, pp. 375–386, April 2004.

[9] H. Zhao, Q. Zheng, W. Zhang, B. Du, and Y. Chen, "A version-aware computation and storage trade-off strategy for multi-version VoD systems in the cloud," in *Proceedings of 2015 IEEE Symposium on Computers and Communication (ISCC)*, July 2015, pp. 943–948.

[10] "MP4Box," 2018. [Online]. Available: https://gpac.wp.imt.fr/mp4box/

[11] "Youtube: Live encoder settings, bitrates, and resolutions." [Online]. Available: https://support.google.com/youtube/answer/2853702?hl=en

[12] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of youtube network traffic at a campus network measurements, models, and implications," *Computer Networks*, vol. 53, no. 4, pp. 501 – 514, 2009.