CABaRet: Leveraging Recommendation Systems for Mobile Edge Caching

Savvas Kastanakis
University of Crete and FORTH, Greece
kastan@csd.uoc.gr

Pavlos Sermpezis
FORTH, Greece
sermpezis@ics.forth.gr

Vasileios Kotronis
FORTH, Greece
vkotronis@ics.forth.gr

Xenofontas Dimitropoulos
University of Crete and FORTH, Greece
fontas@ics.forth.gr

ABSTRACT

Joint caching and recommendation has been recently proposed for increasing the efficiency of mobile edge caching. While previous works assume collaboration between mobile network operators and content providers (who control the recommendation systems), this might be challenging in today's economic ecosystem, with existing protocols and architectures. In this paper, we propose an approach that enables cache-aware recommendations without requiring a network and content provider collaboration. We leverage information provided publicly by the recommendation system, and build a system that provides cache-friendly and high-quality recommendations. We apply our approach to the YouTube service, and conduct measurements on YouTube video recommendations and experiments with video requests, to evaluate the potential gains in the cache hit ratio. Finally, we analytically study the problem of caching optimization under our approach. Our results show that significant caching gains can be achieved in practice; 8 to 10 times increase in the cache hit ratio from cache-aware recommendations, and an extra 2 times increase from caching optimization.

CCS CONCEPTS

• **Networks** → **Network performance evaluation**; **Mobile networks**;

KEYWORDS

Mobile Edge Networks; Recommendation Systems; Joint Caching and Recommendation

1 INTRODUCTION

Mobile Edge Caching (MEC) is one of the key technologies for 5G networks [7] that can reduce latency of service delivery and offload traffic from backhaul links. In MEC, caches are located at the edge of the mobile network (e.g., base stations), and thus have limited capacity and serve small –and frequently changing– user populations [10]. These factors, despite the advances in caching policies [10] or delivery techniques [5], limit the possible gains from MEC: capacity is a tiny fraction of today's content catalogs, and traffic is highly variable; hence, a large number of user requests is for non-cached contents, i.e., not served in the edge.

A recently proposed solution for increasing the efficiency of MEC is jointly caching and recommending content [2, 4, 13]. Recommendation Systems (RS) are integrated in many popular services (e.g., YouTube, Netflix) and significantly affect the user demand [6, 14]. Therefore, steering recommendations towards cached contents, can significantly increase the cache hit ratio, even with small caches or populations.

However, joint caching and recommendation requires collaboration between network operators and Content Providers (CPs). This might be challenging, due to the different scope of these entities, and the constraints of current network protocols and architectures. For example, CPs encrypt traffic (e.g., https) and do not typically share user-related information [11].

To bridge this gap, we propose an approach that is applicable in today's networks: the network operator leverages the information made available by the RS, and, based on this, provides *independently of the CP* high-quality and cache-friendly recommendations that increase the efficiency of MEC. Specifically, we consider the YouTube service, and design a system/application that (i) obtains video relations from the YouTube API, based on which (ii) it builds extended lists of directly and indirectly related videos, and (iii) carefully steers initial recommendations –and thus user demand– towards cached videos. These operations can take place without any tight collaboration with the CP, thus facilitating the application of joint caching and recommendation approaches by network operators (or other entities), without any need for modifications in architectures or protocols.

Our contributions are summarized as follows:

- We propose an approach that enables joint caching and recommendation, without requiring collaboration between network operators and CPs (Sec. 2).
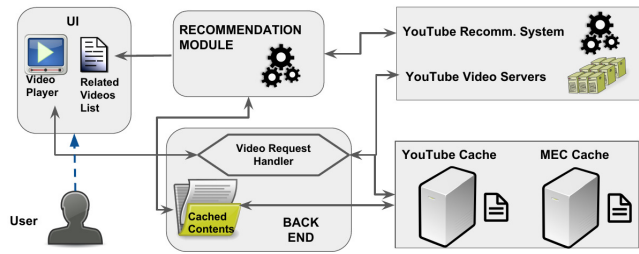
Figure 1: System overview.

- We design an algorithm (named CABaRet) that leverages available information provided by a RS, and returns cache-aware recommendations (Sec. 3).
- We perform extensive measurements over the YouTube service. Our results show that significant caching gains can be achieved in practice; even in conservative scenarios, our approach increases the cache hit ratio by a factor of 8 to 10 (Sec. 4).
- We analytically study the problem of caching optimization under recommendations from CABaRet, and propose an approximation algorithm. We show that when caching is controlled by the network operator, an extra 2 times increase in the cache hit ratio can be achieved (Sec. 5).

Finally, while in this paper we focus on the YouTube service, which provides a public API, our approach can be extended to other video/radio services (e.g., Neflix, Vimeo, Spotify, Pandora) as well, e.g., using offline crawling processes (in case APIs are not available) for discovering content relations.

2 SYSTEM OVERVIEW

The proposed approach can be implemented in a lightweight system/application that runs on mobile devices, and is triggered either by the network operator or by the user. The system is composed of the *user interface* (UI), the *back-end*, and the *recommendation module*, as depicted in Fig. 1.

User Interface (UI). The UI resembles the original content service UI. For instance, in the YouTube case, the UI contains a search bar, a video player, and a list of related videos[1]. The users search, browse, and watch videos through the UI.

Back-end. The back-end is responsible for (i) retrieving the list of cached video IDs (e.g., in the form of a text file), and (ii) streaming videos to the UI. Depending on the scenario, the list of cached video IDs can be already known to the network operator, e.g., in the case of network-controlled caching. Alternatively, they can be requested from the content provider directly[2], or discovered through offline network measurements (e.g., latency [9], or DNS resolution [1]) by the network operator. The video requested by the user is delivered/streamed from the CP's (e.g., YouTube's) origin server, or the CP's cache, or an edge cache. In case the caches are controlled by the content provider (which is the most prevalent scenario today), the user-service communication can be encrypted

---

[1]In the remainder we use the terms *content* and *video* interchangeably.
[2]Note that this refers to an *aggregate* list of the IDs of all the videos stored in a cache, without containing any user information or violating privacy policies.
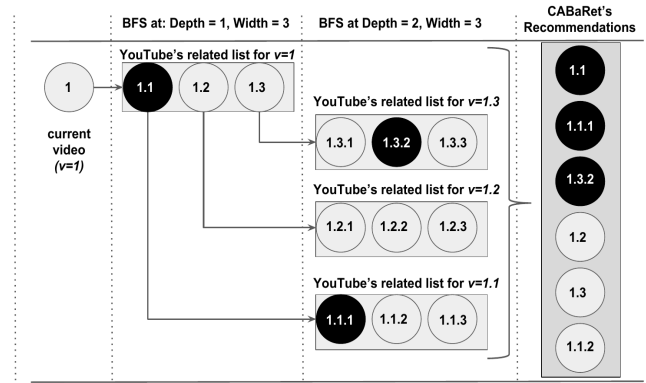
Figure 2: CABaRet: example with $D_{BFS} = 2$, $W_{BFS} = 3$, $N = 6$. Cached videos are denoted with black color.

(e.g., https requests directly to YouTube) and remain transparent to the network operator.

Recommendation Module. The recommendation module is triggered upon each content request, and (i) receives as input the video v that the user currently watches, (ii) retrieves from the YouTube API a list of video IDs directly/indirectly related to v, (iii) extracts from the back-end the list of cached video IDs, and (iv) builds a list of *related and cached* video IDs and recommends it to the user, according to the cache-aware recommendation algorithm of Sec. 3. This recommendation process is lightweight and can return the list of recommendations very fast (e.g., ~1sec. in our prototype), without affecting the user experience.

3 CACHE-AWARE RECOMMENDATIONS

In existing approaches, e.g., [2, 4, 13], the "most related" contents that are also cached, are recommended to users. However, this requires the system to be aware of the *content relations* (e.g., similarity scores, user preferences/history, trending videos), i.e., information owned by the content provider. Such data are unlikely to be disclosed to third parties, due to privacy and/or economic reasons (e.g., advertising).

In our approach, the system leverages information about content relations that is made publicly available by the RS of the content service (i.e., YouTube in this paper). In particular, when a user watches a video v, the system requests from the YouTube API a list of video IDs $\mathcal{L}$ related to v, i.e., the videos that YouTube would recommend to the user. Then it requests the related video IDs for every video in $\mathcal{L}$ and adds them in the end of $\mathcal{L}$, and so on, in a Breadth-First Search (BFS) manner. In the end of the process, the list $\mathcal{L}$ contains IDs of videos directly and indirectly related to v, from which the top N cached and/or highly related to v videos are finally recommended to the user. The list $\mathcal{L}$ is (i) much larger than the list of videos recommended by YouTube, and thus it is more probable to contain cached videos that are related to v, and (ii) built based on video relations provided by YouTube itself, which satisfies a high quality of recommendations.

We detail our recommendation algorithm (CABaRet) in Sec. 3.1, and discuss the related design implications in Sec. 3.2.

3.1 The Recommendation Algorithm

Input. The recommendation algorithm receives as input:

- v: the video ID (or URL) which is currently watched
- N: the number of videos to be recommended
- C: the list with the IDs of the cached videos
- D_{BFS}: the depth to which the BFS proceeds
- W_{BFS}: the number of related videos that are requested *per content* from the YouTube API (i.e., the "width" of BFS)

Output. The recommendation algorithm returns as output:

- $\mathcal{R}$: ordered list of N video IDs to be recommended.

Workflow. CABaRet searches for videos related to video v in a BFS manner as follows (*line 1* in Algorithm 1). Initially, it requests the W_{BFS} videos related to v, and adds them to a list $\mathcal{L}$ in the order they are returned from the YouTube API. For each video in $\mathcal{L}$, it further requests W_{BFS} related videos, as shown in Fig. 2, and adds them in the end of $\mathcal{L}$. It proceeds similarly for the newly added videos, until the depth D_{BFS} is reached; e.g., if $D_{BFS} = 2$, then $\mathcal{L}$ contains W_{BFS} video IDs related to v, and $W_{BFS} \cdot W_{BFS}$ video IDs related to the related videos of v.

Then, CABaRet searches for video IDs in $\mathcal{L}$ that are also included in the list of cached videos C and adds them to the list of video IDs to be recommended $\mathcal{R}$, until all IDs in $\mathcal{L}$ are explored or the list $\mathcal{R}$ contains N video IDs, whichever comes first (*lines 4–9*). If after this step, $\mathcal{R}$ contains less than N video IDs, $N - |\mathcal{R}|$ video IDs from the head of the list $\mathcal{L}$ are added to $\mathcal{R}$; these IDs correspond to the top $N - |\mathcal{R}|$ non-cached videos that are directly related to video v (*lines 10–15*).

Algorithm 1

CABaRet: Cache-Aware & BFS-related Recommendations

$\quad$ *Input* : $v, N, C, D_{BFS}, W_{BFS}$
1: $\mathcal{L} \leftarrow BFS(v, D_{BFS}, W_{BFS})$ $\qquad$ ▷ ordered set of video IDs
2: $\mathcal{R} \leftarrow \emptyset$ $\qquad$ ▷ ordered set of video IDs to be recommended
3: $i \leftarrow 1$
4: **for** $c \in \mathcal{L}$ **do**
5: $\quad$ **if** $i \leq N$ and $c \in C$ **then**
6: $\qquad$ $\mathcal{R}.append(c)$
7: $\qquad$ $i \leftarrow i + 1$
8: $\quad$ **end if**
9: **end for**
10: **for** $c \in \mathcal{L} \setminus \mathcal{R}$ **do**
11: $\quad$ **if** $i \leq N$ **then**
12: $\qquad$ $\mathcal{R}.append(c)$
13: $\qquad$ $i \leftarrow i + 1$
14: $\quad$ **end if**
15: **end for**
16: $return$ $\mathcal{R}$

3.2 Implications and Design Choices

High-quality recommendations. Using the YouTube recommendations ensures strong relations between videos that are directly related to v (i.e., BFS at depth 1). Moreover, while the YouTube RS finds hundreds of videos highly related to v, only a subset of them

are recommended to the user [3]. The rationale behind CABaRet is to explore the related videos that are not communicated to the user. To this end, based on the fact that related videos are similar and have high probability of sharing recommendations (i.e., if video a is related to b, and b to c, then it is probable that c relates to a), CABaRet tries to infer these latent video relations through BFS. Hence, videos found by BFS in depths > 1 are also (indirectly) related to v and probably good recommendations as well.

To further support the above claim, we collect and analyze datasets of related YouTube videos. Specifically, we consider the set of most popular videos, let $\mathcal{P}$, in a region, and for each $v \in \mathcal{P}$ we perform BFS by requesting the list of related videos (similarly to *line 1* in CABaRet). We use as parameters $W_{BFS} = \{10, 20, 50\}$ and $D_{BFS} = 2$, i.e., considering the directly related videos (depth 1) and indirectly related videos with depth 2. We denote as $\mathcal{R}_1(v)$ and $\mathcal{R}_2(v)$, the set of videos found at the first and second depth of the BFS, respectively. We calculate the fraction of the videos in $\mathcal{R}_1(v)$ that are also contained in $\mathcal{R}_2(v)$, i.e., $I(v) = \frac{|\mathcal{R}_1(v) \cap \mathcal{R}_2(v)|}{|\mathcal{R}_1(v)|}$. High values of $I(v)$ indicate a strong similarity of the initial content v with the set of indirectly related contents at depth 2.

Table 1 shows the median values of $I(v)$, over the $|\mathcal{P}| = 50$ most popular contents in the region of Greece (GR), for different BFS widths. As it can be seen, $I(v)$ is very high for most of the initial videos v. For larger values of W_{BFS}, $I(v)$ increases, and when we fully exploit the YouTube API capability, i.e., for W_{BFS}=50, which is the maximum number of related videos returned by the YouTube API, the median value of $I(v)$ becomes larger than 0.9. Finally, we measured the $I(v)$ in other regions as well, and observed that even in large (size/population) regions, the $I(v)$ values remain high, e.g., in the United States (US) region, $I(v)$=0.8 for W_{BFS}=50.

Table 1: $I(v)$ vs. W_{BFS} for the region of GR.

W_{BFS} :	10	20	50
$I(v)$:	0.70	0.85	0.92

Tuning CABaRet. The parameters D_{BFS}, W_{BFS} can be tuned to achieve a desired performance, e.g., in terms of probability of recommending a cached or highly related video.

For large D_{BFS}, the similarity between v and the videos *at the end of the list* $\mathcal{L}$ is expected to weaken, while for small D_{BFS} the list $\mathcal{L}$ is shorter and it is less probable that a cached content is contained in it. Hence, the parameter D_{BFS} can be used to achieve a trade-off between quality of recommendations (small D_{BFS}) and probability of recommending a cached video (large D_{BFS}). The number of related videos requested per content W_{BFS}, can be interpreted similarly to D_{BFS}. A small W_{BFS} leads to considering only top recommendations per video, while a large W_{BFS} leads to a larger list $\mathcal{L}$.

Remark: YouTube imposes quotas on the API requests per application per day, which prevents API users from setting the parameters W_{BFS} and D_{BFS} to arbitrarily large values.

In practice, CABaRet can be fine-tuned through experimentation with real users, e.g., A/B testing iterations, which is a common approach for tuning recommendation systems [3].

4 MEASUREMENTS AND EVALUATION

We conduct measurements and experiments over the YouTube service[3], to investigate the performance (in terms of cache hit ratios) of our approach in MEC scenarios. The setup of the scenarios is presented in Sec. 4.1, and the results in Sec. 4.2.

4.1 Setup

The YouTube API provides a number of functions to retrieve information about videos, channels, user ratings, etc. In our measurements, we request the following information:

- the most popular videos in a region (max. 50)
- the list of related videos (max. 50) for a given video

Remark: In the remainder, we present results collected during March 2018, for the region of Greece (GR). Nevertheless, our insights hold also in the other regions we tested (e.g., US)..

Caching. We assume a MEC cache storing the most popular videos in a region. We populate the list of cached contents with the top C video IDs returned from the YouTube API.

Recommendations. We consider two classes of scenarios with (i) YouTube and (ii) CABaRet recommendations. In both cases, when a user enters the UI, the 50 most popular videos in her region are recommended to her (as in YouTube's front page). Upon watching a video v, a list of $N = 20$ videos is recommended to the user; the list is (i) composed of the top N directly related videos returned from the YouTube API (YouTube scenarios), or (ii) generated by CABaRet with parameters N, W_{BFS} and D_{BFS} (CABaRet scenarios).

Video Demand. In each experiment, we assume a user that enters the UI and watches one of the initially recommended (i.e., 50 most popular) videos at random. Then, the system recommends a list of N videos $(r_1, r_2, ..., r_N)$, and the user selects with probability p_i to watch r_i next. We set the probabilities p_i to depend on the order of appearance –and not the content– and consider *uniform* $(p_i = \frac{1}{N})$ and *Zipf* $(p_i \sim \frac{1}{i^\alpha})$ scenarios; the higher the exponent α of the Zipf distribution, the more preference is given by the user to the top recommendations (user preference to top recommendations has been observed in YouTube traffic [9]).

4.2 Results

4.2.1 Single Requests.

We first consider scenarios of single requests (similarly to [2, 13]). In each experiment i ($i = 1, ..., M$) a user watches one of the top popular videos, let $v_1(i)$, and then follows a recommendation and watches a video $v_2(i)$. We measure the Cache Hit Ratio (CHR), which we define as the fraction of the *second requests* of a user that are for a cached video (since the first request is always for a cached –top popular– video):

$$CHR = \frac{1}{M} \cdot \sum_{i=1}^{M} \mathbb{I}_{v_2(i) \in C}$$

where $\mathbb{I}_{v_2(i) \in C} = 1$ if $v_2(i) \in C$ and 0 otherwise, and M the number of experiments[4].

---

[3]Our experiments and use of the YouTube API conform to the YouTube terms of service https://www.youtube.com/static?template=terms.
[4]We considered all possible experiments on the collected dataset.
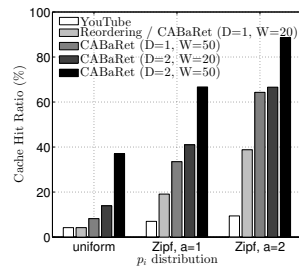

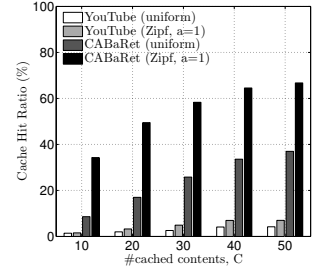**Figure 3: CHR under different BFS parameters.**

Figure 4: CHR vs. # cached contents C
(W_{BFS}=50, D_{BFS}=2).

CHR vs. BFS parameters. Fig. 3 shows the CHR achieved by CABaRet under various parameters, along with the CHR under regular YouTube recommendations, when caching all the most popular videos ($|C|$=50). The efficiency of caching significantly increases with CABaRet, even when only directly related contents are recommended (D_{BFS}=1), i.e., *without loss in recommendation quality*. Just reordering the list of YouTube recommendations (as suggested in [9]), brings gains when p_i is not uniformly distributed. However, the added gains by our approach are significantly higher. As expected, the CHR increases for larger W_{BFS} and/or D_{BFS}; e.g., CABaRet for W_{BFS}=50 and D_{BFS}=2, achieves 8-10 times higher CHR than regular YouTube recommendations. Also, the CHR increases for more skewed p_i distributions, since top recommendations are preferred and CABaRet places cached contents at the top of the recommendation list.

In experiments concerning the –larger– US region, the CHR values are lower for both regular YouTube ($< 0.5\%$) and CABaRet ($1\% - 43\%$) recommendations, due to the fact that the top popular videos appear with lower frequency in the related lists. However, the *relative gains* from CABaRet are consistent with (or even higher than) the presented results.

CHR vs. number of cached videos. We further consider scenarios with varying number of cached contents $C = |C|$. In each scenario, we assume that the C most popular contents are cached. Fig. 4 shows the CHR achieved by CABaRet, in comparison to scenarios under regular YouTube recommendations. The results are consistent for all considered values of C; the CHR under CABaRet is significantly higher than in the YouTube case. Moreover, even when caching a small subset of the most popular videos, CABaRet brings significant gains. E.g., by caching $C = 10$ out of the 50 top related contents CABaRet increases the CHR from 2% and 3.2% to 17% and 50%, for the uniform and Zipf(α=1) scenarios, respectively.

4.2.2 Sequential Requests.

We now test the performance of our approach in scenarios where users enter the system and watch a sequence of K, $K > 2$, videos (similarly to [4], and in contrast to the previous case, where they watch only two videos, i.e., $K = 2$). At each step, the system recommends a list of videos to the user by applying CABaRet on the currently watched video. We denote as $v_k(i)$ the kth video requested/watched by a user in experiment i. We measure the CHR,
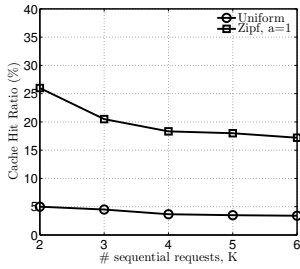
Figure 5: CHR vs. # requests in sequence K (C=20, W_{BFS}=20, D_{BFS}=2). Note: y-axis up to 40%.
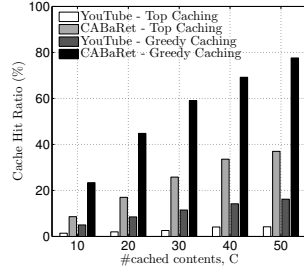
Figure 6: CHR vs. # cached contents (p_i uniform, W_{BFS}=50, D_{BFS}=2).

which is now defined as

$$CHR = \frac{1}{M} \cdot \sum_{i=1}^{M} \sum_{k=2}^{K} \mathbb{I}_{v_k(i) \in C}$$

where $\mathbb{I}_{v_k(i) \in C} = 1$ if $v_k(i) \in C$ and 0 otherwise, over $M = 100$ experiments per scenario.

Moving "farther" from the initially requested video (which belongs to the list of most popular and cached videos) through a sequence of requests, we expect the CHR to decrease, due to lower similarity of the requested and cached videos. However, as Fig. 5 shows, the decrease in the CHR (under CABaRet recommendations) is not large. The CHR remains close to the case of single requests (i.e., for K=2 in the x-axis), indicating that our approach performs well even when we are several steps far from the cached videos. In fact, caching more than the top most popular videos appearing on the front page, would further reduce the CHR decrease.

5 CACHING OPTIMIZATION

In this section, we extend our study by considering the scenario where CABaRet and the caches are controlled by the same entity, e.g., the network operator. Network operator-controlled caching is the most commonly considered scenario in related work for MEC; although in most of today's architectures the caches are actually controlled by the CP.

In this scenario, the network operator can optimize caching decisions, thus further increasing the efficiency of CABaRet recommendations. Note that still there is no need for collaboration between the operator and the CP (e.g., possessing full knowledge of the RS), assumed in previous works [2, 4, 13].

In the following, we first analytically formulate and study the problem of optimizing the caching policy, and propose an approximation algorithm with provable performance guarantees (Sec. 5.1). We then evaluate the performance of this *joint caching and recommendation* approach (Sec. 5.2).

5.1 Optimization Problem & Algorithm

Let a content catalog $\mathcal{V}$, $V = |\mathcal{V}|$, and a content popularity vector $\mathbf{q} = [q_1, ..., q_V]^T$. Let $\mathcal{L}(v) \subseteq \mathcal{V}$ be the set of contents that are explored by CABaRet (at *line 1*) for a content $v \in \mathcal{V}$, and denote $\mathcal{L} = \bigcup_{v \in \mathcal{V}} \mathcal{L}(v)$.

For some set of cached contents $C \subseteq \mathcal{V}$, and a content v, CABaRet returns a list of recommendations $\mathcal{R}(v)$ ($|\mathcal{R}(v)| = N$), in which at most N contents $c \in C \cap \mathcal{L}(v)$ appear at the top of the list. Therefore, CHR can be expressed as

$$CHR(C) = \sum_{v \in \mathcal{V}} q_v \sum_{i=1}^{N(C,v)} p_i \qquad (1)$$

where $N(C, v) = \min\{|C \cap \mathcal{L}(v)|, N\}$, and p_i is the probability for a user to select the ith recommended content.

Then, the problem of optimizing the caching policy (to be jointly used with CABaRet), is formulated as follows:

$$max_C \ CHR(C) \qquad s.t., |C| \leq C \qquad (2)$$

where C is the capacity of the −MEC− cache. We prove the following for the optimization problem of Eq. (2).

LEMMA 1. *The optimization problem of Eq. (2): (i) is NP-hard, (ii) cannot be approximated within $1 - \frac{1}{e} + o(1)$ in polynomial time, and (iii) has a monotone (non-decreasing) submodular objective function, and is subject to a cardinality constraint.*

PROOF. Items (i) and (ii) of the above lemma, are proven by reduction to the *maximum set coverage* problem, and we prove item (iii) using standard methods (see, e.g., [5, 13]). The detailed proof is omitted due to space limitations. □

If we design a greedy algorithm that starts from an empty set of cached contents $C_g = \emptyset$, and at each iteration it augments the set C_g (until $|C_g| = C$) as follows:

$$C_g \leftarrow C_g \cup \arg \max_{v \in \mathcal{V}} CHR(C_g \cup \{v\}), \qquad (3)$$

then the properties stated in item (iii) satisfy that it holds [8]

$$CHR(C_g) \geq \left(1 - \frac{1}{e}\right) \cdot CHR(C^*) \qquad (4)$$

where C^* the optimal solution of the problem of Eq. (2).

Remark: While Eq. (4) gives a lower bound for the performance of the greedy algorithm, in practice greedy algorithms have been shown to perform often very close to the optimal.

5.2 Evaluation of Greedy Caching

Calculating the CHR from Eq. (1) requires running a BFS (CABaRet, *line 1*) and generating the lists $\mathcal{L}(v)$, for every content $v \in \mathcal{V}$. In practice, for scalability reasons, the most popular contents (i.e., with high q_i) can be considered by the greedy algorithm in the calculation of the objective function Eq. (1), since those contribute more to the objective function. However, any video in the catalog is still candidate to be cached, e.g., a video with low q_i can bring a large increase in the CHR through its association with many popular contents.

In fact, in our experiments, for the calculation of Eq. (1), we consider only the 50 most popular videos, for which we set $q_i = \frac{1}{50}$. Nevertheless, in the different scenarios we tested, only 10% to 30% of the cached videos (selected by the greedy algorithm) were also in the top 50 most popular.

In Fig. 6, we compare the achieved CHR when the cache is populated according to the greedy algorithm of Eq. (3) (*Greedy Caching*), and with the top most popular videos (*Top Caching*). *Greedy caching*

always outperforms *top caching*, with an increase in the CHR of around a factor of 2 for uniform video selection (for the Zipf(a=1) scenarios we tested, the CHR values are even higher, and the relative performance is 1.5 times higher). This clearly demonstrates that the gains from joint recommendation and caching [2, 13], are applicable even in simple practical scenarios (e.g., CABaRet & greedy caching). Finally, while *greedy caching* increases the CHR even with regular YouTube recommendations, the CHR is still less than 50% of the CABaRet case with *top caching*. This further stresses the benefits from CABaRet's cache-aware recommendations.

6 RELATED WORK

In the recent works of [2, 4, 13], caching and recommendation of contents is considered for mobile networks. [2, 13] jointly optimize the caching and recommendation policies, while [4] optimizes cache-aware recommendations for sequential requests. Our work is complementary to these works: we propose an approach for deploying joint caching and recommendation in practice, without collaboration between network operators and content providers. Moreover, our extensive measurements over the real YouTube service show that the gains demonstrated in [2, 4, 13] can be achieved even today, using a simple and lightweight system.

[9] studies how to improve the efficiency of YouTube caches, by reordering the list of recommendations. CABaRet goes beyond reordering, and leverages more information that is available through the YouTube API; as our results show, this can bring substantial gains in the CHR.

Finally, research on the performance of the YouTube CDN in relation to network parameters [1, 9, 12], indicates that our approach can be relevant to wireline networks as well. For instance, on top of classic load-balancing, *network-aware recommendations* (e.g., taking into account routing policies [1, 12], inter-domain load balancing [12], cache locations [1]) can be used to improve the user QoE (e.g., latency).

7 CONCLUSION

In this paper, we proposed an approach that enables joint caching and recommendation in MEC, without requiring tight collaboration of network and content providers. We considered the YouTube service as our use case, and showed that significant gains can be achieved by leveraging available information from its recommendation system.

The effect of geographical and temporal factors on the gains from cache-aware recommendations, can be further investigated in the future to extend our initial experimental results. Moreover, we believe that analytically studying the interplay between content relations and potential gains with CABaRet, as well as extending our approach to other online content services, are interesting research directions.

ACKNOWLEDGEMENTS

REFERENCES

[1] V.K. Adhikari, S. Jain, and Z.-L. Zhang. 2011. Where do you "tube"? uncovering YouTube server selection strategy. In *Proc. IEEE ICCCN*.
[2] L.-E. Chatzieleftheriou, M. Karaliopoulos, and I Koutsopoulos. 2017. Caching-aware Recommendations: Nudging User Preferences towards better Caching Performance. In *Proc. IEEE INFOCOM*.
[3] P. Covington, J. Adams, and E. Sargin. 2016. Deep neural networks for YouTube recommendations. In *Proc. ACM RecSys*.
[4] T. Giannakas, P. Sermpezis, and T. Spyropoulos. 2018. Show me the Cache: Optimizing Cache-Friendly Recommendations for Sequential Content Access. In *Proc. IEEE WoWMoM*.
[5] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire. 2012. FemtoCaching: Wireless video content delivery through distributed caching helpers. In *Proc. IEEE INFOCOM*.
[6] C. Gomez-Uribe and N. Hunt. 2016. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2016), 13.
[7] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young. 2016. Mobile Edge Computing A key technology towards 5G. *ETSI White Paper No. 11* (2016).
[8] A. Krause and D. Golovin. 2012. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems* 3, 19 (2012), 8.
[9] D.K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen. 2015. Cache-centric video recommendation: an approach to improve the efficiency of YouTube caches. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 11, 4 (2015), 48.
[10] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas. 2016. Placing Dynamic Content in Caches with Small Population. In *Proc. IEEE INFOCOM*.
[11] J. Leguay, G. Paschos, E. Quaglia, and B. Smyth. 2017. Cryptocache: Network caching with confidentiality. In *Proc. IEEE ICC*.
[12] R. Mok, V. Bajpai, A. Dhamdhere, and kc. Claffy. 2018. Revealing the Load-Balancing Behavior of YouTube Traffic on Interdomain Links. In *Proc. PAM*.
[13] P. Sermpezis, T. Spyropoulos, L. Vigneri, and T. Giannakas. 2017. Femto-Caching with Soft Cache Hits: Improving Performance with Related Content Recommendation. In *Proc. IEEE GLOBECOM*.
[14] R. Zhou, S. Khemmarat, and L. Gao. 2010. The Impact of YouTube Recommendation System on Video Views. In *Proc. ACM IMC*.