CLoSER: Video Caching in Small-Cell Edge Networks with Local Content Sharing

Shadab Mahboob[a], Koushik Kar[1a], Jacob Chakareski[b], Md Ibrahim Alam[a]

[a]*Rensselaer Polytechnic Institute, Troy, NY 12180. Email: shadab.mahboob2018@gmail.com, koushik@ecse.rpi.edu, alamm2@rpi.edu.*
[b]*New Jersey Instititute of Technology, Newark, NJ 07102. Email: jacob.chakareski@njit.edu.*

Abstract

We consider the problem of video caching in an edge network consisting of a set of small-cell base stations (SBS) that can share content among themselves over a high-capacity short-delay local network, or fetch the videos from a remote server over a long-delay connection. Even though the problem of minimizing the overall video playout delay in our framework is NP-hard, we develop CLoSER, an algorithm that can efficiently compute a solution that is close to the optimal, where the degree of sub-optimality depends on the worst case video-to-cache size ratio. In comparison with related prior work on video caching and streaming, CLoSER specifically focuses on the benefits of local sharing of the initial portion of the video content in reducing the video playout delay, and provides strong optimality guarantees with a low-complexity algorithm. We extend CLoSER to an online setting where the video popularities are not known a priori but are estimated over time through a limited amount of periodic information sharing between SBSs. With such online video popularity estimation, a distributed implementation of CLoSER requires zero explicit coordination between the SBSs and runs in $O(NK + K \log K)$ time, where N is the number of SBSs (caches) and K the maximum number of videos. We carry out simulations using YouTube and Netflix video request traces, as well as synthesized traces with the same marginal distributions as the traces but varying degree of temporal correlations, and demonstrate that our algorithm uses the SBS caches effectively to reduce the video delivery delay and conserve the remote server's bandwidth. We also show that it outperforms two other reference caching methods adapted to our system setting, over a wide range of remote-to-local bandwidth ratios. Further, we show how CLoSER extends to the scenario where each video may need to be cached in multiple bit-rates, as in Available Bit Rate (ABR) streaming.

Keywords: Collaborative Caching, Video Streaming, Edge Networks.

[1]Corresponding author

1. Introduction

In recent years, the Internet has witnessed tremendous growth in video traffic, as well as in the number and variety of video streaming applications [1]. Applications such as YouTube, Netflix, Amazon Prime Video, Hulu, and Sling TV are contributing to a large part of our daily Internet bandwidth consumption. Live video streaming and video-on-demand (VoD) services are growing, sharing of video news and messages through social networking applications like Facebook and WhatsApp is increasing steadily, and news readers are increasingly utilizing video feeds for their daily news. The growth in video traffic in recent years has also been accompanied by significant changes in video access patterns. Firstly, video quality and bit rate are steadily increasing, with the growing availability of Ultra High Definition (UHD) or 4K video streams that occupy more than double the HD video bit rate. Secondly, the difference between 'live' and 'stored' video is blurring, as more users expect VoD capability for TV shows and news feeds. Nevertheless, the access patterns of such content are often highly correlated temporally, i.e., close to each other in time, after being posted online. Finally, there is considerable growth in video viewing over wireless, over both WiFi and cellular technologies. Increasing deployment of super-fast access technologies such as 802.11ac/802.11ax and 5G is accelerating this growth. There is also increased video viewing on mobile devices such as smartphones and tablets, which are expected to contribute to about 50% of the Internet traffic in a few years [1].

These trends motivate the importance of caching videos close to users to reduce access delay and network/server congestion. Increasing deployment of 5G small-cell systems is expected over the next decade, making *Small-cell Base Stations (SBS)* natural candidates for hosting such caches. The range (coverage area) of these SBSs, and the cache sizes that can be included in them, are expected to be small. At the same time, in many of the deployment scenarios (airports, malls, office buildings, campuses, etc.), such SBSs may be deployed in large numbers and be connected with each other over a fast local area network such as high-speed Ethernet or fiber-optic ring. This motivates pooling resources of multiple such SBSs and using them to collaboratively cache videos for access by users in the entire coverage area.

In this paper, we consider the problem of video caching in a wireless edge network comprising of multiple small-cell base stations that are connected to each other over a high-capacity low-delay local area network. The SBSs host small video caches and can share videos with each other over the local network, or they can fetch videos from a remote server over a long-delay Internet path. We specifically focus on the question of minimizing the *playout delay*, which is becoming increasingly important as that typical video lengths and watch times continue to decrease steadily [2, 3]. For example, as of July 2022, videos that are under 1 minute or less made up 57% of YouTube views, compared to just 11% two years ago [2]. This is driven by reduced user attention spans, more video browsing due to the deluge of user-created video content becoming available, and aggressive video-based marketing on YouTube and other online social platforms like Tiktok and Instagram [4]. Increased video bit-rates (in UHD standards like 4K and 8K, which grew between 2017 and

2022 at an cumulative annual growth rate (CAGR) of about 30% [5]) implies that the time to download the videos (or even an initial few secs of it) can be substantial, emphasizing the importance of local caching. For longer video-streams like movies, where the overall streaming experience may be impacted more significantly by other Quality of Experience (QoE) metrics [6], reducing the playout delay by caching the initial part of the video can still make the user experience smoother.

In this setup, the problem of minimizing the overall video playout delay is NP-hard due to packing-type integrality constraints; even if the integrality constraints are relaxed, the problem is a *concave minimization* problem which is difficult to solve in general. Despite these challenges, we utilize the specific structure of our problem to develop an efficient algorithm, *CLoSER*, that computes a close-to-optimal solution, where the degree of sub-optimality depends on the worst case video-to-cache size ratio. We also extend this algorithm to a dynamic setting where the video popularities are not known a priori but are estimated over time. With such online video popularity estimation, a distributed implementation of CLoSER *does not require any explicit coordination between the SBSs*, as long as an ordering (tie-breaking) rule between the caches and the videos is pre-determined, and information on the video requests from users is periodically shared between the SBSs. This distributed implementation of CLoSER runs in $O(NK + K \log K)$ time, where N is the number of SBSs (caches) and K is the maximum number of videos.

We show via numerical experiments on a small number of caches that our algorithm approaches the optimal (integral) caching solution when the cache size is large relative to the individual video sizes. Simulations conducted on real video access traces (YouTube as well as Netflix video requests) demonstrate the performance trade-offs between the video playout delay and the local/remote bandwidth used, and examine the impact of popularity estimation parameters and re-optimization intervals on the overall caching performance of the system. Exploiting the temporal correlation in video access patterns across the caches, our algorithm is able to effectively use video sharing across the SBS cache pool to reduce the video playout delay and congestion at the remote server. We also carry out simulations on synthesized traces with the same marginal distributions as the real traces but varying degree of temporal correlations, and demonstrate the impact of temporal correlations on the performance of CLoSER. We further show that CLoSER outperforms two other reference caching methods (adapted to our system setting) over a wide range of remote-to-local bandwidth ratios. Finally, we extend and evaluate CLoSER in the scenario where each video may need to be cached in multiple bit-rates, as in Available Bit Rate (ABR) streaming, and compare with the non-ABR case.

2. Comparison with Related Work

The literature on caching in content-centric networks, and even on caching in wireless edge networks, is quite voluminous. While much of the early work on content caching at the wireless edge focuses on optimizing content in a single cache, we will specifically focus on multi-cell collaborative caching strategies, as they are most closely related to our work. For broader surveys of video caching and streaming methods proposed in the literature, see [7, 8].

A number of cooperative caching strategies for cellular networks have been proposed using primarily optimization approaches, considering different objective functions such as overall delay, cost and revenue, together with cache capacity constraints. In [9], the authors maximize a total reward objective for a service provider in a collaborative manner for both coded and uncoded data. In a follow-up study, joint caching, routing, and channel selection is investigated using large-scale column generation optimization [10]. In [11], the aggregated storage and download cost for caching is minimized for both limited and unlimited caching spaces. In [12], the authors utilize Bayesian Learning and recommender systems to predict and utilize personalized preferences in caching. In [13], the authors consider time-scale differences between video caching and retrieval, and propose a two-phase approach to minimize content access delay, even though the solution requires solving a binary integer linear program (ILP). The work in [14] presents a solution to the joint caching and routing in multi-cell multi-access edge networks, specifically for VR videos that are coded in 'tiles'. In [15], the authors propose a two-stage optimization approach for robust caching, and radio and computational resource allocation across multiple cells. In [16], caching is considered jointly with transmission and interference management strategies in the edge, with the video files being encoded into segments. An auction based approach to video caching in a multi-tier system consisting of small-cells, a mobile network operator, and multiple video service providers is presented in [17]. In [18], the content caching problem at the 5G edge is posed and analyzed as a Stackelberg game between the multiple mobile network operators and content providers. In [19], the authors consider QoE aware video content caching across a set of edge servers, and propose an approximation algorithm for minimizing the overall system cost posed as an integer linear program. The authors in [20] pose the content caching problem across multiple wireless edge caching stations in a stochastic optimization framework, and propose a Lyapunov drift based push-and-pop strategy towards minimizing long-term content downloading time. Mobility and contact duration aware content placement in cellular edge networks is considered in [21], for which greedy and genetic algorithm based heuristic methods are proposed and evaluated. The impact of caching and scheduling decisions on quality of experience in many of these methods can be characterized using analytical approaches [22, 23].

Our work differs from prior models and results in this context in the following key aspects. Firstly, unlike the existing literature (including the prior papers mentioned above), we assume *sharing of video content between the SBSs*. More specifically, this amounts to storing (*without duplication*) some of the frequently accessed video content in a group of small local caches (hosted at the SBSs), which share the content between themselves over a high-capacity short-delay local network. Our analysis shows that this sharing is most effective when done for videos that are "moderately popular", i.e., their popularity lies in between the most popular videos (which must be cached at all SBSs) and unpopular videos (which may not be cached at all). For large video files, this sharing may be limited to the initial few seconds or minutes of the video – but may have a significant impact in reducing the playout delay, as will be demonstrated through our evaluation studies. In a recent work [24], the problem of optimal content placement with local content sharing between caches has been considered in a different context, namely between cooperative Internet Service Providers (ISPs). Moreover, while [24] explores the benefits of cooperation on the aggregate ISP utility subject to fairness and reciprocity constraints, our goal is to design efficient content placement algorithms under local sharing, focusing specifically on the video playout delay.

Secondly, our focus is on developing decentralized solutions that *guarantee optimality and yet can be implemented with very low message complexity.* In contrast, many of the prior studies develop complex solutions that may require a centralized coordinator (such as a macro base station or a cloud computation facility), or develop heuristic distributed solutions that do not provide any optimality guarantees.

Our problem is also loosely related to that of device-to-device (D2D) or peer-to-peer (P2P) content caching and delivery in wireless edge networks [25, 26, 27, 28]. However, some of the key performance considerations of content sharing between devices, such a energy efficiency and incentive design – are quite different from playout delay, the key concern in video streaming applications. Due to the large data volume of emerging video content, the practicality of video caching and sharing across energy-constrained edge devices remains questionable. Moreover, the problem formulation we consider in this work is quite different from those that have been considered in other analytical studies of D2D content sharing, such as [27].

As observed in numerous prior papers (see [29], for example), caching of content (including video) can be formalized as a weighted knapsack type problem [30], which is typically NP-hard due to the arbitrary sizes of the content items. The complexity of the knapsack packing constraint goes away when the item sizes are much smaller than the size of the sack (cache), a reasonable assumption in caching, particularly since large videos are broken into smaller units, such as temporal segments (chunks) for ABR streaming [31, 32] or spatio-temporal tiles for $360°$ video streaming [33, 34] over DASH [35]. When considering content sharing among local caches, however, there is an additional level of complexity involved: the caching problem in that case represents a concave minimization [36] problem, which must be solved in a decentralized manner with low coordination complexity. This is a unique technical aspect of our model in which it differs from existing work on video caching.

In recent years, there has been a surge in the use of learning techniques for content caching [37, 38, 12, 39, 40]. In our work, we follow the more traditional approach of separating the popularity estimation and caching optimization steps. While our caching optimization step requires sophisticated analysis, for video popularity estimation we use a very simple form of learning based on exponentially weighted averaging of historical request data, which we found to be quite effective. This simplicity in the popularity estimation process is also motivated by recent observations [41, 42] that simple linear popularity prediction models tend to work well for content caching when using historical data.

3. Model and Formulation

We consider an edge network comprised of a set of N SBSs, indexed as $i = 1, \cdots, N$. There is a set of K videos, indexed as $k = 1, \cdots, K$, which may be downloaded from one or more remote servers. SBS i is associated with cache space C_i, which it uses to selectively cache some of the videos. The SBSs are connected to a high-speed local network over which they can exchange videos with each other. Each end-user is assumed to be associated with one of the SBSs at any given time, although that association may change over time due to user mobility. If the requested video is available at the SBS (cache) that the user is associated with, the video is served to the user with minimal playout delay.[2] Otherwise, the video is either obtained from one of the other SBSs in the local network, or is downloaded from the remote server(s) if it is not available in the local caches. If the video is present in one of the other local SBS caches, we assume an average playout delay of d; otherwise (i.e., if the video is to be fetched from the remote server(s)), the average playout delay is $D > d$.[3] In the rest of this paper, the term 'delay' refers to video playout delay, unless mentioned otherwise. Our system model is illustrated in Figure 1.

Let the size and popularity of video k be represented by s_k and π_k, respectively. The size s_k represents the size of the video, or part of it, that will need to be cached. For short videos, it may represent the entire video. For long videos, it may correspond to the initial few seconds to minutes of the video that is needed for reducing the playout delay (latency); the rest of the video maybe streamed directly from the remote server. If the video is broken up into smaller units (such as tiles in VR applications), then the size s_k and popularity π_k would need to be associated with each such unit in which the video needs to be cached.

Note that the video popularity is considered independent of the SBS (cache) k. Since users will typically be mobile across the coverage area of the SBSs, we reasonably assume that the same popularity vector (which represents the access rates of the videos across the entire population of served users) would apply to all SBSs. Given $\pi = (\pi_k, k = 1, \cdots, K)$, and letting $x_{i,k}$ be a binary variable indicating if video k is cached at SBS i, our goal is to minimize the overall video playout delay, expressed as $\sum_k \pi_k \Big[\sum_i \Big((\max_{i'} x_{i',k} - x_{i,k})\, d \; + (1 - \max_{i'} x_{i',k})\, D \Big) \Big]$.

Note that $(\max_{i'} x_{i',k} - x_{i,k})$ equals 1 only when video k is cached locally (in one of the SBS caches) but not at cache i, and zero otherwise. Therefore, the term $(\max_{i'} x_{i',k} - x_{i,k})\, d$ represents the additional video delay incurred if requested video k is not cached at SBS i but has to be fetched from one of the other SBSs. On

[2]For ease of exposition, we take this delay to be zero. There is no loss of generality here, as assuming that this delay on an average is a positive number δ does not affect our algorithm or its underlying analysis.

[3]Note that we consider the video *playout* delay and not the video delivery delay, and therefore this delay is assumed independent of the video size. The video playout delay, one of the most important factors affecting QoE, is primarily a function of the quality of the connection (path) (such as bandwidth, round trip time) over which the video is delivered.

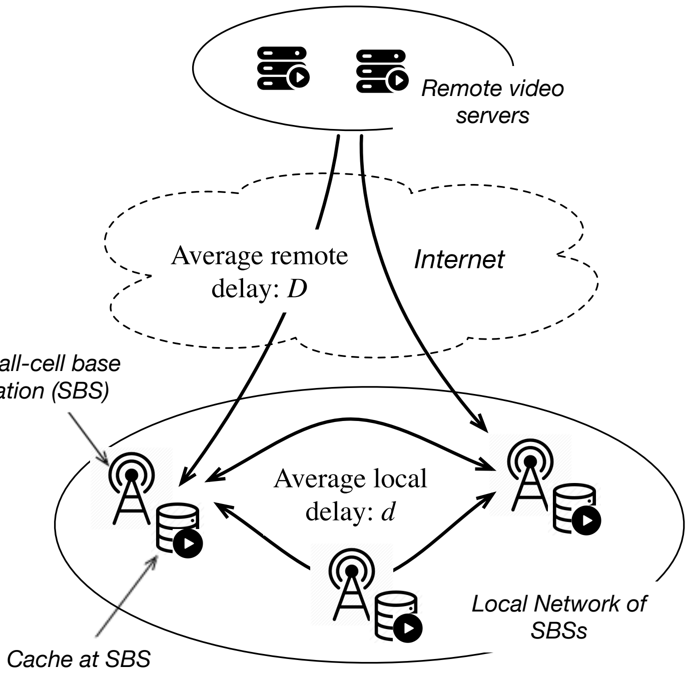

Figure 1: Illustration of our System Model.

the other hand, the term $(1 - \max_{i'} x_{i',k})$ is 1 only if video k is not cached locally at all (and therefore must be fetched from a remote server incurring an additional average delay cost of D), and zero otherwise. Since the above objective is equivalent to maximizing $\sum_k \pi_k \left[d \sum_i x_{i,k} \ + \ (D-d) \sum_i \max_{i'} x_{i',k} \right]$, we define our *Mininum Playout-delay Edge Caching (MPEC)* question as the following constrained maximization problem:

$$\text{maximize} \ \sum_k \pi_k \left[d \sum_i x_{i,k} \ + \ (D-d) \sum_i \max_{i'} x_{i',k} \right], \tag{1}$$

$$\text{subject to} \quad \textstyle\sum_k x_{i,k} s_k \le \ C_i, \quad \forall i, \tag{2}$$

$$x_{i,k} \in \ \{0,1\}, \qquad \forall i, \forall k. \tag{3}$$

Next we mention a few points about the structure and complexity of the MPEC problem formulated in (1)-(3), towards motivating the solution approach that we will present in the next section. The complexity of the problem comes from two aspects: (a) the non-linearity (non-convexity) of the objective function (1); and (b) the integrality (binary nature) of the optimization variables in (3). The constraints (2)-(3) represents integral "packing type" constraints which make the problem NP-hard. For a single SBS, the problem reduces to the 0-1 knapsack problem [30]. While the binary knapsack problem is NP-hard, is can be solved in pseudo-polynomial time using dynamic programming; several heuristic approaches are also known to work well in practice. However, most of these approaches are not easily amenable to distributed implementation with low coordination and message exchange complexity, which is highly desirable in our setting.

Towards addressing (a), we note that since $D > d$, when the integrality constraints (3) are relaxed, the MPEC problem posed in (1)-(3) represents a *concave minimization* problem over a polyhedral set [36]. While such problems are NP-hard in general, the max terms in the objective function can be replaced by a set of linear terms and additional linear constraints. However, this not only results in additional variables and constraints, it does not help towards developing distributed solutions with low coordination message complexity.

In our algorithm that we describe next (in Section 4), both (a) and (b) are addressed, and the resulting solution closely approximates the optimal integral solution of the problem, when the SBS cache capacities are sufficiently large compared to the individual video sizes (or units in which the videos are cached).[4] Further, the algorithm can be implemented is a distributed manner with *no explicit coordination* between the SBSs, as long as a tie-breaking rule is agreed upon in advance, and information about video requests from users are periodically shared between SBSs to enable estimation of the popularity vector π.

4. Algorithm and Analysis

At a high level, the algorithm we develop – ***CLoSER**: **E**dge **C**aching with **Lo**cal **S**haring* attempts to use sharing of videos across the local caches at SBSs towards reducing the average playout delay, as quantified in (1). If a popular video cannot be found at an SBS, it is accessed at one of the other SBSs (if available) before requesting it directly from the remote server. This not only reduces the playout delay but also reduces the backhaul bandwidth and congestion at the remote video servers (see Figure 1) at the cost of more bandwidth use in the local (higher-capacity) network.

The basic idea behind CLoSER is simple. The “highly popular” videos are cached in every SBS, while the “unpopular” videos are not cached locally. The videos that are “moderately popular” are cached at only one or more (but perhaps not all) of the SBSs in the local network, and that determination is based on the video popularity and size, the sizes of the SBS caches, and the average local and remote delays. This results in an algorithm with two to three stages (phases), as we describe next.

Towards developing intuition behind the algorithm and its optimality analysis, we first present the for the special case of unit size videos, with the cache sizes (possibly different across caches) being integral multiples of this unit video size. This special case admits a simpler algorithm that can be described and illustrated easily, as well as a simpler optimality proof that still captures the core essence of the argument. We then

[4]If video sizes are large (like HD/UHD-quality movies), caches may store the beginning few seconds (minutes) of each video, seeking to reduce the initial playout delay, while the rest of the video is streamed directly from the server to the user. In other cases where the video size is large, such as in 360-degree videos, the video can be cached in units of small-size tiles or collection of tiles in the most popular view [43]. Therefore, our assumption on the ratio of the video caching unit to the cache size being small generally holds true.

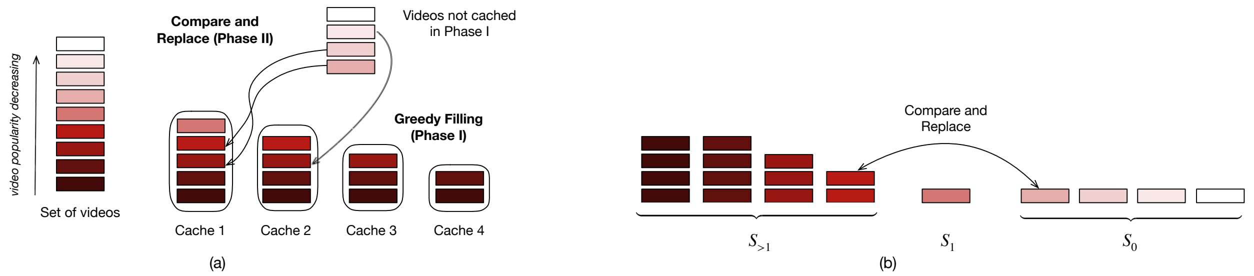

Figure 2: Illustration of CLoSER for unit video sizes.

extend this algorithm to the general case of arbitrary video sizes (in addition to cache sizes being arbitrary), and prove that it is optimal in an approximate (asymptotic) sense.

4.1. CLoSER for Unit Video Sizes

In this special case, $s_k = 1\ \forall k$; also, $C_i = m_i$ for some positive integer $m_i,\ \forall i$. This implies that there is no loss due to fragmentation of the videos while caching. The formulation of MPEC remains the same as that in (1)-(3) except that (2) is reduces to $\sum_{k\in\mathcal{K}} x_{i,k} \leq\ C_i,\ \forall i$.

Without loss of generality, assume that the videos are indexed in a non-increasing order of their popularity values (ties broken arbitrarily), i.e., $\pi_k \geq \pi_{k'}$ when $k < k'$. Let the number of cached copies of video k during the run of the algorithm be denoted by n_k. The algorithm works in two phases, as described next.

```
Algorithm 1: CLoSER: Unit Video Sizes (Input: $d, D, \pi$)
  Phase I (Greedy Filling):
  for each SBS (cache) $i$ do
      Fill up cache $i$ with videos $1, 2, \cdots, m_i$.
  for each video $k$ do
      $n_k = \{i|k \leq m_i\}$ // number of cached copies of $k$.
  Set $S_{>1} = \{k|n_k > 1\}, S_1 = \{k|n_k = 1\}, S_0 = \{k|n_k = 0\}$.
  Phase II (Compare and Replace):
  Let $k_1$ = last index in $S_{>1}$; $k_2$ = first index in $S_0$.
  while ($S_{>1}$ and $S_0$ are both non-empty) && ($\frac{\pi_{k_2}}{\pi_{k_1}} > \frac{d}{ND-(N-1)d}$) do
      Replace any one copy of video $k_1$ with $k_2$, and set $n_{k_1} = n_{k_1} - 1$ and $n_{k_2} = 1$.
      Set $S_0 = S_0 - \{k_2\}$, $S_1 = S_1 + \{k_2\}$, and $k_2 = k_2 + 1$.
      if ($n_{k_1} == 1$) then
          Set $S_{>1} = S_{>1} - \{k_1\}; S_1 = S_1 + \{k_1\}$, and $k_1 = k_1 - 1$.
```

Phase I (Greedy Filling): In this phase, each cache is filled up greedily and independently with videos in the order $1, 2, 3, \cdots$ (non-increasing order of popularity) up to the cache size limit. For the case of unit video sizes, this simply means that cache i caches up to video with index $C_i = m_i$. Note that at the end of this phase, the highly popular videos may be cached in multiple (possibly all) caches, whereas videos with low popularity may not be cached at all. In general, a video k is cached in any cache i such that $k \leq m_i$. Let

$S_{>1}, S_1$, and S_0 respectively denote the sets of videos that have been cached in multiple caches, in a single cache, or not cached at all. The greedy filling step is illustrated in Figure 2(a).

Phase II (Compare and Replace): In the second phase, the n_k values (and accordingly, the $S_{>1}, S_1$, and S_0 sets) are altered as some of the videos from the set S_0 are cached in one of the caches (and therefore move to set S_1) replacing videos that are present in more than one cache. This replacement happens if two videos $k_1 \in S_{>1}$ and $k_2 \in S_0$ satisfy a "popularity ratio test": $\pi_{k_2}/\pi_{k_1} > d/(ND - (N-1)d)$. Thus, in this phase, the n_k values of the videos in $S_{>1}$ may reduce, and the set $S_{>1}$ may shrink as well, as some of the videos in $S_{>1}$ may just have one copy left and therefore move to the set S_1. Further, no video in S_0 or S_1 is cached multiple times in this phase, so no video is added to $S_{>1}$. The set S_0 may shrink as well, as some of its videos may be cached (in a single cache at most) and therefore move to S_1. The set S_1 can only expand, due to videos from both $S_{>1}$ and S_0 possibly moving to this set. See the illustration in Figure 2(b). The popularity ratio test is conducted by picking k_1 to be the video with the largest index in $S_{>1}$ and k_2 as the video with the smallest index in S_0; therefore the replacement step can be viewed as the boundary between S_1 and S_0 moving right by one step; and the boundary between $S_{>1}$ and S_1 staying the same, or moving left by one step. The algorithm is described in Algorithm 1. Note that this compare and replace phase stops (and the algorithm ends) when the k_1, k_2 thus picked fails the ratio test, or either $S_{>1}$ or S_0 becomes empty (i.e., k_1 becomes 0 or k_2 becomes $N+1$).

The algorithm requires initial sorting of the videos according to their π_k values, which takes $O(K \log K)$ time. Once the videos are sorted, Phase I takes $O(k)$ per SBS, or $O(NK)$ in total. Before Phase II, constituting the sets $S_{>1}, S_1, S_0$ takes $O(NK)$ time. Phase II involves $O(K)$ compare and replace steps. Therefore, the overall time complexity of CLoSER for unit size videos is $O(NK + K \log K)$.

Our optimality claim for unit video sizes is stated in Theorem 1 (proof in Appendix).

Theorem 1. *If $s_k = 1$ for all videos k, and all cache sizes C_i are integral, then Algorithm 1 computes an optimal solution of MPEC as posed in (1)-(3).*

4.2. CLoSER for Arbitrary Video Sizes

For arbitrary video sizes, CLoSER (Edge Caching with Local Sharing) works along similar lines as Algorithm 1, but with the following differences. First, in both Phase I (Greedy Filling) and Phase II (Compare and Replace), we ignore the integrality constraints, and allow videos to be filled or replaced *fractionally* if needed, so that no cache space is wasted. Second, popularity values π_k are replaced by *popularity density* values $w_k = \pi_k/s_k$, and the video indices are sorted in an non-increasing order of the w_k values, i.e., $w_k \geq w_{k'}$ when $k < k'$. Last, there is a final Rounding phase (Phase III) of the algorithm where the space allocated to videos in S_1 are reallocated to the same videos, but in a way that satisfies the integrality constraints. At the same time, any fractional videos included in $S_{>1}$ are dropped. Therefore, the rounding step ensures that the final solution contains only full videos. Further details on the three phases of the algorithm is provided next, and the CLoSER algorithm for this general case is fully described in Algorithm 2.

Phase I (Greedy Filling (*fractional*)): Note that in this phase, cache i stores up to video m_i, where m_i satisfies $\sum_{k=1}^{m_i-1} s_k < C_i$ and $\sum_{k=1}^{m_i} s_k \geq C_i$. Thus videos $1, \cdots, m_i - 1$ are fully included, but only a part of video m_i may be included to fill up the cache space.

Phase II (Compare and Replace (*fractional*)): In this phase, we replace n_k by y_k, the aggregate size (including fractional) of video k as cached across all SBSs. Thus, if video k is cached fully in only one cache, $y_k = s_k$. The sets $S_{>1}, S_1$, and S_0 are defined accordingly, as shown in Algorithm 2. In Phase II, "extra" copies of video $k_1 \in S_{>1}$ are replaced by video $k_2 \in S_0$ (even though this replacement may only be fractional), if the popularity density values of videos k_1, k_2 satisfy a ratio test, as described in the algorithm.

Phase III (Rounding): The rounding phase collects all the space allocated to videos in S_1 across the different caches, and reallocates those videos in that space sequentially. Note that Phase II may have split the single copy of a video k (belonging to S_1) across multiple caches, and the rounding step collects all that and uses it to place video k in a single cache. At any point the next cache among those that have available space (given by the set R) is chosen. If this cache (say i) does not have enough remaining space to accommodate the video (say κ), i.e., $c_i < s_\kappa$, then video κ is dropped. This ensures only full size videos in S_1 remain in the final solution. Any fractional videos included in $S_{>1}$ and S_0 are also dropped. Therefore, the rounding phase ensures that the integrality constraints (3) are satisfied by the final solution.

The complexity of Phases I and II in Algorithm 2 is similar to those of Algorithm 1. The Rounding phase takes an additional $O(N + K)$ time. Therefore, the overall time complexity of CLoSER remains the same as that in the special case, i.e., $O(NK + K \log K)$.

Our optimality claim for the general case (arbitrary video and cache sizes) is stated in Theorem 2 (proof in Appendix).

Theorem 2. *Let $\epsilon = \frac{\max_k s_k}{\min_i C_i}$. Then Algorithm 2 computes a feasible solution to MPEC as posed in (1)-(3), with an objective function value no less than $(1 - O(\epsilon))$ of the optimum.*

From Theorem 2, we note that the approximation factor $(1 - O(\epsilon))$, which approaches 1 when the cache sizes are sufficiently large compared to the individual video sizes (or the units in which the videos are cached), which is a reasonable assumption in practice.

4.3. Distributed and Online Implementation

Algorithm 2 can be easily implemented in a distributed manner provided the SBSs (caches) agree upon: (i) The video popularity vector π, and how ties are broken when sorting the videos according to their popularity values; (ii) An ordering (tie-breaking rule) between the caches. The video popularity vector will

Algorithm 2: CLoSER: *Arbitrary Video Sizes* (**Input:** d, D, π, s)

```
Define $w_k = \pi_k / s_k \ \forall k$, and reorder video indices $k$ in non-increasing order of $w_k$ values.
Phase I (Greedy Filling (fractional)):
for each SBS (cache) $i$ do
    Fill up cache $i$ with videos $1, 2, \cdots, m_i - 1$ (full), and $m_i$ (possibly fractional).
for each video $k$ do
    $y_k$ = aggregate size of all cached copies of $k$ (including fractional).
Set $S_{>1} = \{k | y_k > s_k\}, S_1 = \{k | y_k = s_k\}, S_0 = \{k | y_k < s_k\}$.
Phase II (Compare and Replace (fractional)):
Let $k_1$ = last index in $S_{>1}$; $k_2$ = first index in $S_0$.
while ($S_{>1}$ and $S_0$ are both non-empty) && ($\frac{w_{k_2}}{w_{k_1}} > \frac{d}{ND-(N-1)d}$) do
    Replace an amount $r$ of video $k_1$ (from any cache(s) containing $k_1$) with $k_2$, where
      $r = \min\{y_{k_1} - s_{k_1}, s_{k_2} - y_{k_2}\}$. Set $y_{k_1} = y_{k_1} - r$ and $y_{k_2} = y_{k_2} + r$.
    if ($y_{k_1} == s_{k_1}$) then
        Set $S_{>1} = S_{>1} - \{k_1\}; S_1 = S_1 + \{k_1\}$, and $k_1 = k_1 - 1$.
    if ($y_{k_2} == s_{k_2}$) then
        Set $S_0 = S_0 - \{k_2\}; S_1 = S_1 + \{k_2\}$, and $k_2 = k_2 + 1$.
Phase III (Rounding):
Remove any fractional videos from $S_{>1}$ and $S_0$.
Let $c_i$ = the amount of space allocated to videos in $S_1$ in cache $i$ after Phase II. Set $R = \{i | c_i > 0\}$.
Let $\kappa_1$ ($\kappa_2$) be the first (last) video in $S_1$.
while ($\kappa_1 \le \kappa_2$) do
    Pick the first cache $i$ in $R$.
    if ($c_i > s_{\kappa_1}$) then
        Cache video $\kappa_1$ in $i$, and set $c_i = c_i - s_{\kappa_1}$.
    else
        $R = R - \{i\}$.
    $\kappa_1 = \kappa_1 + 1$.
```

typically be estimated based on the user video requests at different caches. If the request data is periodically shared between caches, this estimation can be done independently at each cache. Any ties in the π_k values can be broken according to some predetermined rule, such as video id. The caches can be indexed simply in the decreasing order of their cache sizes, with ties broken according to any pre-determined manner.

Note that Phase I (Greedy Filling) is a fully decentralized process (requiring no coordination) which caches can carry out fully independently of each other. Phases II and III may appear centralized as they consider the set of all caches at the same time; however, under the conditions (i) and (ii) specified above, the caches can carry out the computations in Phases II and III without any coordination with each other. Each cache will calculate the same overall solution from Phases II and III, but will only cache the videos that it itself is supposed to cache. In other words, to implement Algorithm 2 no explicit coordination or messaging is required between caches. Further, after the completion of the algorithm, each cache will also automatically know which videos are being cached in the other caches. This helps in requesting videos from those caches, as needed in a practical (online) implementation of the algorithm, as we describe next. Note that since Phase I can be run in parallel across the different caches, the time complexity is reduced from $O(NK)$ to $O(K)$. However, the overall time complexity of the distributed implementation of CLoSER remains the same as the centralized one, which is $O(NK + K\log K)$.

In practice, the video population would not be fixed, and the relative popularities of the videos will also evolve over time. To adapt our algorithm to such scenarios, we can re-estimate the video popularity values periodically, and re-optimize the caching solution accordingly. In our performance evaluation as described in Section 5, we re-estimate and popularity vector π periodically, by counting the number of videos requested by users in the local network (across all SBSs) over a time window W ending at the current time. This requires the SBSs (caches) *to share with each other the list of videos requested, but no other information exchange or coordination is required.* Each cache then applies Exponential Weighted Moving Average (EWMA) to calculate the the popularity values to be used in the caching re-optimization. More specifically, the popularity of video k at the beginning of the t^{th} window, π_k^t, is calculated as

$$\pi_k^t = (1-\alpha)\pi_k^{t-1} + \alpha\frac{n_k^{t-1}}{W}, \tag{4}$$

where n_k^t represents the number of video requests during the $(t-1)^{\text{th}}$ window (across all SBSs), and α, where $0 \leq \alpha \leq 1$ is an appropriately chosen weighting parameter.

Note that CLoSER needs to be re-run (and a new optimal solution needs to be re-computed) at the end of each window, assuming that the popularity vector π has changed significantly from last time. CLoSER then reoptimizes the caching solution, which happens in the background. In this reoptimization process, CLoSER tries to minimize the use of remote bandwidth: thus, if a cache needs to obtain a new video due to this reoptimization, it preferentially gets it from another cache, and contacts the server only if the video is not stored in the local network. We observe (see Section 5) that the amount of bandwidth consumed (both locally and remotely) is only a very small fraction of the total bandwidth consumed, implying that this reoptimization process has very limited overhead.

4.3.1. Extensions and Limitations

While our system model described in Section 3 was kept simple for ease of exposition, our algorithm and analysis generalizes easily (at least in theory) in several directions, some of which are outlined below. Extending and evaluating CLoSER to these more general models/contexts is however left for future work.

First, note that in the objective function of (1), the delays d and D are independent of the video k. In practice, as each video may be associated with different resolutions, bit-rates etc., these delays could potentially depend on the video k. This could be easily incorporated into our framework. In this case, each video k is associated with a local delay d_k and remote delay D_k (assumed to be known or estimated a priori), and we are seeking to maximize

$$\sum_k \pi_k \left[d_k \sum_i x_{i,k} \; + \; (D_k - d_k) \sum_i \max_{i'} x_{i',k} \right]$$

subject to the same constraints as before, given by (2)-(3). To accommodate this generalization, the only change needed in Algorithms 1 and 2 is a modification to the "popularity ratio test" condition: $\pi_{k_2}/\pi_{k_1} > d_{k_1}/(ND_{k_2} - (N-1)d_{k_2})$ (Algorithm 1) and $w_{k_2}/w_{k_1} > d_{k_1}/(ND_{k_2} - (N-1)d_{k_2})$ (Algorithm 2). It is easy to verify that all the proofs work out as is with this change, and Theorems 1 and 2 still hold, irrespective of whether the delays d_k, D_k depend on the video size s_k or not.

Second, the above generalization is also important in extending our model and solution to ABR video [31, 32]. In ABR streaming, supported by DASH [35], since video is coded at multiple quality levels, caching of a video may also need to be done at multiple levels. Given that quality fluctuations have a very detrimental effect on the user-perceived QoE [6], it is desirable that the initial part of the video is cached at all the levels at which the users are likely request it. This can be done by considering each quality level as a different video k in our system model, with its corresponding π_k, s_k, d_k and D_k values. Accommodating multiple (quality) representations of a video in the cache will obviously require larger cache sizes to attain the same performance as the non-ABR case. The extension of CLoSER to ABR settings is evaluated in Section 5.7.

Third, in applying our solution to multiview or 360° videos (which are typically coded in units of tiles) [33], the natural question that arises is at what granularity the video should be cached: tiles or views. We believe that videos should be cached in terms of views – each of which corresponds to a collection of tiles – with which the popularity indices are directly associated with. While caching in units of tiles can allow space savings by considering the overlaps between views, missing tiles create a very undesirable viewing experience. In other words, views must be streamed to the user with the full set of spatial tiles constituting the view, which justifies using views as the unit of caching for multiview videos.

Finally, it may be worth clarifying that CLoSER only targets minimizing the playout delay, which is only one of the metrics impacting the overall viewing experience (QoE) of the user. Maximizing QoE over the entire video streaming period is an orthogonal problem that requires different optimization formulations and techniques. In the context of ABR video, for example, the QoE over the entire duration of the video requires a broader range of strategies involving pre-fetching of video content and dynamic adaptation of video quality. Further, it requires consideration of the tradeoffs between the different QoE metrics such as average quality levels versus degree of quality level fluctuations [6]. Caching can still be very effective in improving quality over the entire streaming period, by allowing more proactive buffering of video content (while the video is being streamed) to improve overall video quality level, reduce the frequency or amplitude of quality level fluctuations. This may require dynamic monitoring (predicting) of the available network bandwidth, and involve proactive requesting of the quality layers depending on the available cache space and the predicted network bandwidth, among other parameters.

5. Performance Evaluation

Simulations using real video request data traces from YouTube [44] and Netflix [45] were conducted to evaluate CLoSER's performance. The nature of the results between the two traces were mostly similar. In some cases, we present the results for both datasets to illustrate any differences between the performances of the two datasets; in other cases, as approriate, results are presented for only one of the datasets.

5.1. Datasets

Netflix Dataset: The publicly available Netflix Prize dataset [45] is a trace of about 15.5 million video requests from a database of 17770 videos. These traces were collected between the years 1999 to 2005. From the dataset, the request pattern for the most popular 3000 videos is extracted and used in simulations. The video size information is not provided in the dataset; further, the Netflix videos are expected to be large and not all of it needs to be cached in an SBS for smooth streaming. Hence, to attain the goal of CLoSER (minimizing the playout delay only) only the initial part of the video is cached, which we assume to be uniformly distributed between 10 MB and 20 MB. At current full-HD quality (1080p) Netflix video data rates of about 50 MB per min [46], this corresponds to caching the initial 12–24 secs of video playing time.

YouTube Dataset: The publicly available YouTube video request dataset [44] contains view statistics and metadata of 1 million YouTube videos collected in 2013. We consider the top 1420 videos from this YouTube dataset, as the popularity values beyond that is too low for the video to be cached. The YouTube dataset is quite large, and to reduce simulation time we downsample the number of views by 50:1, which generates a total of 8.9 million requests. Thus 50 consecutive views of a video in the original data trace are represented

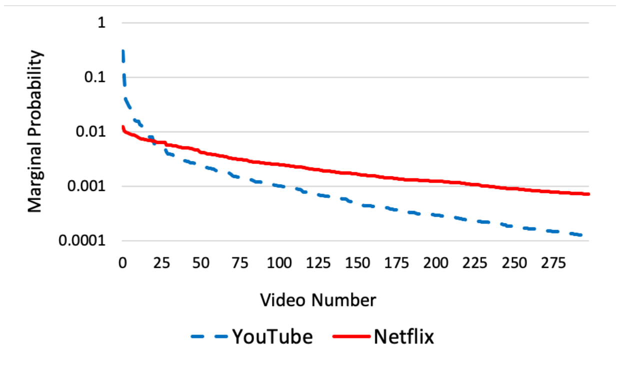

Figure 3: Probability distribution of YouTube and Netflix datasets (semi-log scale).

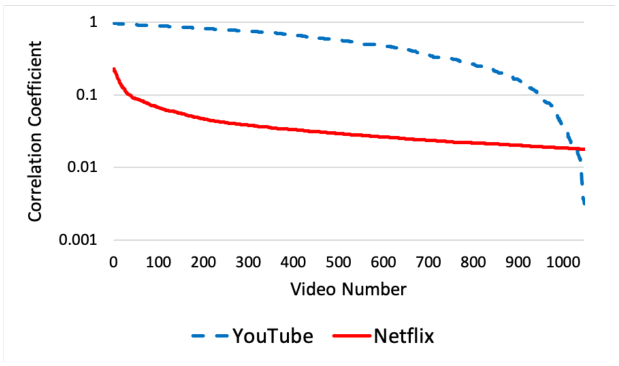

Figure 4: Correlation coefficient distribution of YouTube and Netflix datasets (semi-log scale).

by 1 video request in our trace, whose request time is set to the average of the requesting times of the 50 videos in the original trace that it represents. Note that this downsampling does not significantly change the statistics or the temporal characteristics of the distribution. The downsampling creates a trace that is of the same order (in terms of size) to the Netflix dataset discussed earlier. The YouTube dataset does not provide the size of the videos, but records the watch times. These watch times are fairly small – with an average 2.4 minutes and maximum of 13 minutes. Therefore, we set the part of each video to be cached to be the average watch time of the video, and the actual size is calculated assuming a 20 MB per min data rate, which roughly corresponds to the current HD quality (720p) data rates of YouTube videos [47].

While we have experimented with different values (ratios) of playout delays d and D, the results presented in this paper are mostly for $d = 500$ milliseconds (ms), remote delay $D = 5$ seconds (s). At Netflix HD quality video bit rates of 50 MB per min, for example, this may allow up to 7.5 secs of video playout buffering. However, when the time in between requesting the video and starting to receive it (which is a function of the generally unknown processing and propagation delays in the local or remote network) is considered, this playout buffering time would reduce by a few secs, but would still be reasonable.

Figure 3 shows the probability of occurrence (i.e., popularity values) for the top 275 videos in the YouTube and Netflix datasets, sorted in the decreasing order of these popularity values. We observe that compared to Netflix, the popularity distribution of YouTube videos is more skewed, i.e., YouTube has a few videos that are very popular, while the rest have low popularity. This maybe partly due to the more "viral" nature of a small fraction of the YouTube videos, and the fact that the YouTube data trace was collected over a shorter period of time. Further, to understand the differences between the two datasets in terms of temporal correlation, we calculated the *correlation coefficient* for each video k in the trace, measured as the probability of requesting video k again immediately after it has been requested. The correlation coefficient for the top 1050 videos (in the decreasing order of the correlation coefficient) in the Netflix and YouTube datasets are shown in Figure 4. As expected, the correlation coefficients for the top few Youtube videos are quite high,

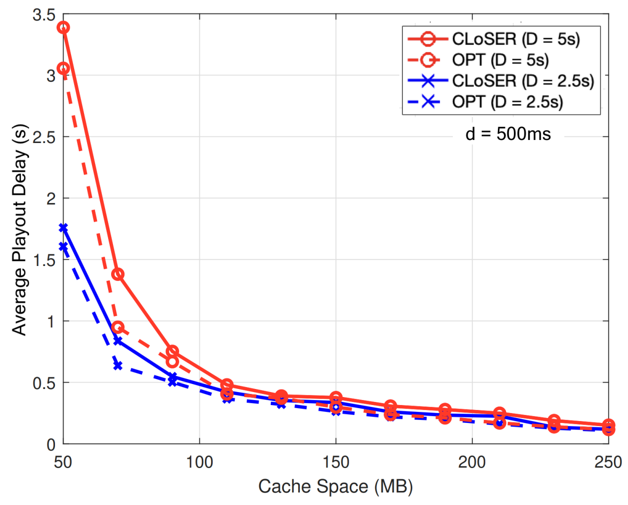

Figure 5: Avg. Playout Delay vs Cache Size per SBS (YouTube).

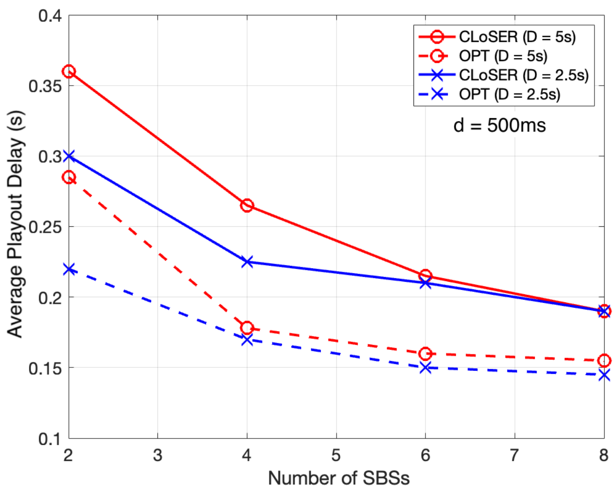

Figure 6: Avg. Playout Delay vs No. of SBSs (YouTube).

indicating that they are requested repeatedly quite a few times before another video is requested. For Netflix videos, the correlation coefficient is more uniformly distributed, implying that the video requests are more randomized compared to the YouTube trace.

5.2. Comparison with the Integral Optimal Solution

We first compare CLoSER with the optimal solution (OPT), where OPT is calculated by solving MPEC (as defined by (1)-(3)) exactly. Due to the integrality constraints, solving OPT is computationally intensive; therefore, this comparison study is limited to the 20 highest-popularity videos in the dataset. Figure 5 shows the average delay by varying cache size for each SBS from 50 MB to 250 MB with a fixed number of SBSs, $N = 4$ for the YouTube dataset. For the same dataset, Figure 6 shows the average delay by varying the number of SBSs from 2 to 8 when the size of each cache is fixed at 300 MB.

We can observe from Figure 5 that CLoSER closely approximates OPT in average playout delay at all cache size values. Note that when the cache space is very small, very few videos can be accommodated in the cache, and neither CLoSER nor OPT performs very well. On the other hand, for sufficiently large cache spaces, the entire video set can be stored in each cache (or at least in the local network), and both approaches perform very well. This explains why the performance of CLoSER and OPT are close towards these two extremes. From Figure 6, the average delays under CLoSER and OPT tend to get closer with increase in number of SBSs, as we intuitively expect as well. From Figures 5 and 6, it can be observed that even in the worst case (within the range of the settings simulated), the average playout delay under CLoSER is well within a factor of 2 of that of OPT. The figure shows that even with a modest cache size of 300 MB per SBS, the benefits of local sharing taper off around $N = 8$ SBSs (caches). Since the range of a 5G base station is expected to be in the range between 50m and 500m (depending on the exact frequency and technology used) [48], this could amount to local sharing among SBSs that span an academic or residential area to an entire small university campus.

We next report the performance results of the online implementation of CLoSER in terms of (i) average fetched bytes and (ii) average playout delay. The average fetched bytes, which represent the total amount of bytes used by CLoSER in the local or remote network to serve a video request on an average, consist of two parts: (a) *Delivery bytes*: The bytes actually fetched from the remote server, or from another cache (SBS) in the local network, at the time of serving a video request; (b) *Reoptimization (Reopt) bytes*: The bytes transferred over the local network or over the link to the remote server(s) during the reoptimization process under CLoSER, which happens periodically (end of each time window of length W).

5.3. Delivery-vs-Remote Bytes and Local-vs-Remote Bytes

Next we present the performance results of CLoSER characterized by the delivery versus reoptimization bytes, each further sub-divided based on whether the bytes were fetched locally (from another SBS cache) or from the remote server. Note that a higher Delivery-to-Reopt bytes ratio and a higher Local-to-Remote bytes ratio would imply better performance.

Figures 7, 8, and Tables 1, 2 provide the values of average delivery and reoptimization bytes per request for both YouTube and Netflix traces with $\alpha = 0.4$ and $W = 30{,}000$. The figures show the changes in average bytes with the progression of window numbers, while the Tables provide the average values over the whole window numbers. The fetched bytes are differentiated based on whether they were fetched over the local network (i.e., from another SBS cache) or from the remote server. We assume a cold start, i.e., the caches do not have any videos cached initially. As expected, as the window number progresses, some of the requested videos get cached, and popularity (π_k) estimates improve, which results in a drop in remote delivery bytes. Simultaneously, the local delivery bytes increases, as most of the requested videos are fetched from a cache in the local network (instead of being fetched from the server). The average delivery bytes (both Local and Remote) for both datasets supports the observation, i.e., averaging over the entire trace duration, the local delivery bytes dominates the remote delivery bytes.

From the tables we observe that Netflix trace yields better results in terms of Remote-to-Total bytes fetched when compared to the YouTube trace. The reason behind is that we observed the local delivery bytes for YouTube trace to increase slowly until the end of the simulation period. This is due to the fact that the YouTube trace is too short to attain a steady state in that period. Generally, YouTube videos are posted with a much higher frequency and new videos are posted all the time. This implies that it may take the system much longer to reach a steady state, or a steady state may not be reached at all. This is unlike our results for Netflix data trace, where the requests come with less temporal correlation and sudden surges in request pattern is less likely, thus allowing a steady state to be reached more quickly.

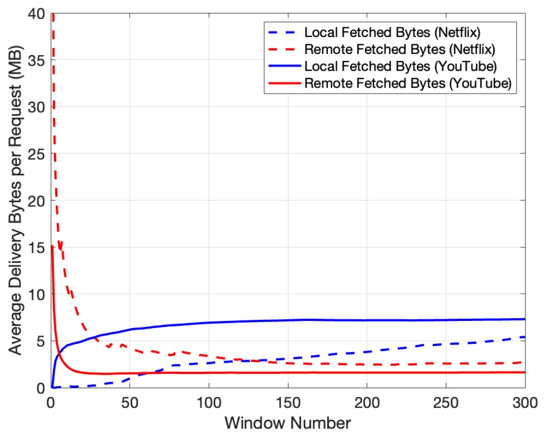

Figure 7: Average Delay Bytes per Request.

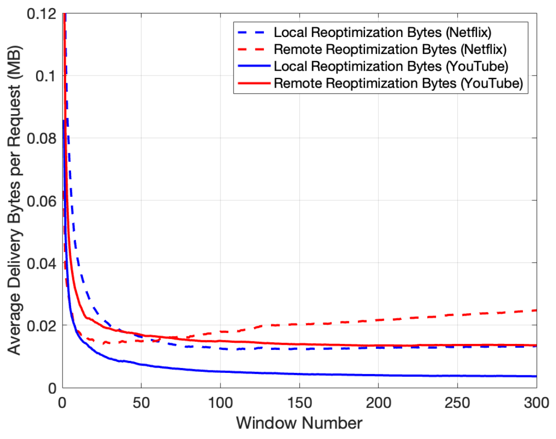

Figure 8: Average Reoptimization Bytes per Request.

	Local	Remote	Remote/Total (%)
Avg. Delivery Bytes (MB)	3.9	2.99	43.4
Avg. Reopt Bytes (MB)	0.0164	0.02	54.9
Reopt Bytes/Total (%)	0.42	0.66	

Table 1: Avg. Delivery and Reopt Bytes (YouTube)

	Local	Remote	Remote/Total (%)
Avg. Delivery Bytes (MB)	6.72	1.72	20.4
Avg. Reopt Bytes (MB)	0.0162	0.0062	27.7
Reopt Bytes/Total (%)	0.24	0.36	

Table 2: Avg. Delivery and Reopt Bytes (Netflix)

The reopt bytes shows similar trend as the delivery bytes. Over the time it tends to saturate for Netflix but not saturating for YouTube. It is important to note that the reopt bytes are *two orders of magnitude smaller* than the delivery bytes, which implies that the overhead of this reoptimization process is negligible. In other words, almost all of the fetched bytes are utilized directly in serving the video requests. This is certainly a desirable feature and implies that the reoptimization process that is carried out in the background consumes very little additional bandwidth (both local and remote), even though it is important for adapting the solution to changes in the video popularity.

5.4. Effect of Reoptimization Window and Temporal Correlation

This section investigates the effect of the window length parameter W, i.e., the number of requests after which the video popularity estimates are recalculated, and the cache contents are reoptimized. From (4), we note that reducing α has an effect similar to that of increasing W. Our experiments seemed to indicate that varying W (for a fixed α) provides greater control (on delay and bandwidth consumed) over varying α (for a fixed W). Therefore, in the results shown below, we fix $\alpha = 0.4$, and vary the window length W. Note

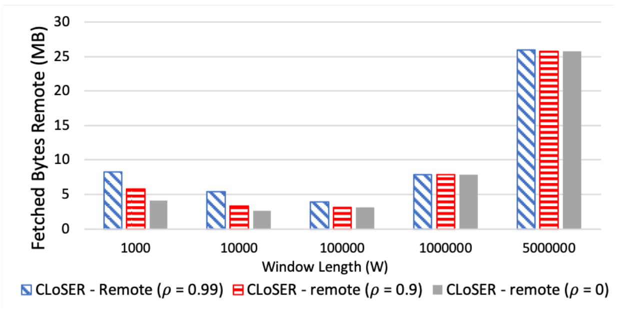

Figure 9: Fetched Bytes - Remote for different ρ (Youtube).

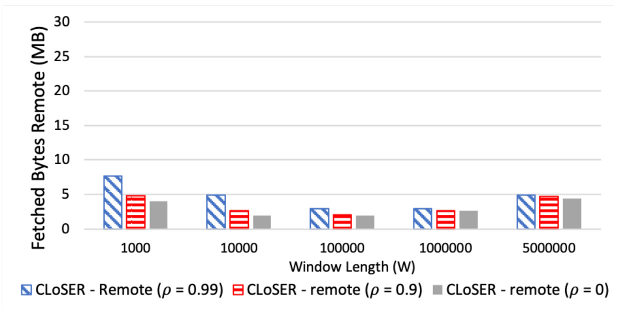

Figure 10: Fetched Bytes - Remote for different ρ (Netflix).

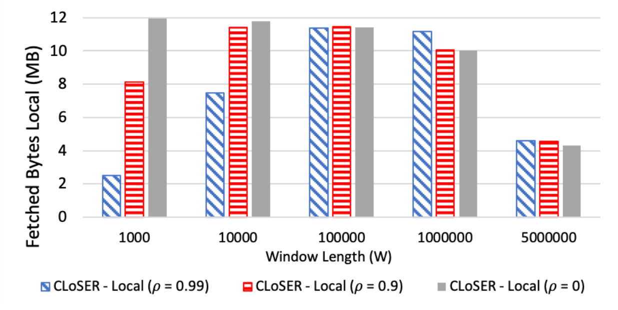

Figure 11: Fetched Bytes - Local for different ρ (Youtube).

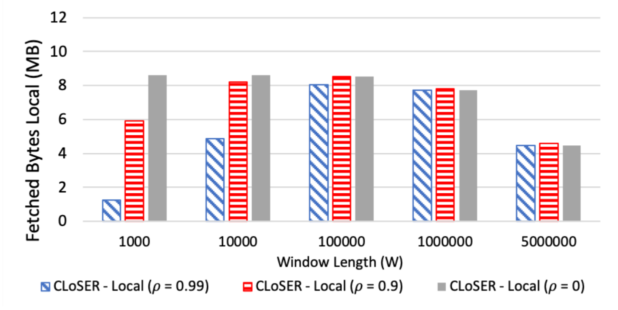

Figure 12: Fetched Bytes - Local for different ρ (Netflix).

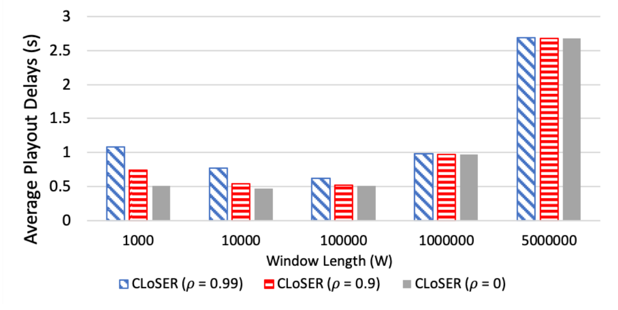

Figure 13: Average Playout Delay for different ρ (Youtube).

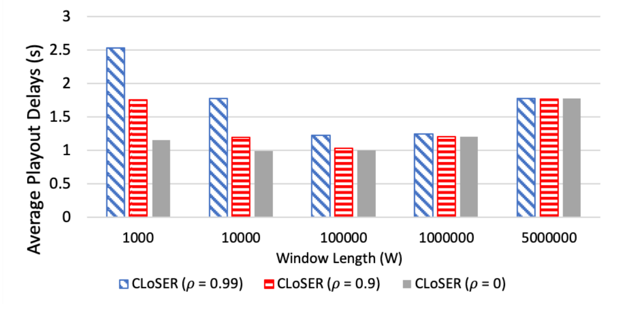

Figure 14: Average Playout Delay for different ρ (Netflix).

that the window length is measured in terms of the total number of requests (across all SBSs), although it could be equivalently represented in terms of per-SBS requests, or in time units as well.

We expect that the best caching performance will be attained at an intermediate range of values of W. When the window size W is very small, then the estimated π_k values do not reflect the true popularity values of the videos; therefore, the caching performance is expected to improve with increasing W. On the other hand, when the window size is very large, any dynamic change in the true popularity values is not immediately reflected in the estimated π_k values. In practice, the right window length W can be determined by analysis of the timescale at which the video popularities change.

Note that the results so far are for two specific traces (YouTube and Netflix), so there is limited room to experiment with other request patterns. Caching performance can be affected by the degree of temporal correlation in the request patterns, and we expect that the choice of the estimation (reoptimization) window can have a significant impact on this performance as well. Towards that end, we generate traces with the same marginal distribution as the Netflix and YouTube video traces, but with different levels of temporal correlation in the request pattern, parameterized by ρ (correlation parameter) defined similarly to the correlation parameter utilized in Figure 4. More specifically, with probability ρ, a request generated is for the same video that was requested last, and with the probability $(1-\rho)$, the requested video is chosen randomly from all K videos according to their marginal distributions. Thus $\rho = 0$ corresponds to i.i.d. request generation, and the degree of temporal correlation increases with $\rho \in [0, 1)$. Figures 9-14 show our performance metrics (remote and local fetched bytes, and average delay) for three different values of ρ, namely $\rho = 0, 0.9, 0.99$ for a wide range of the window size parameter, W, from 1000 requests to 5 million (around 30% and 50% of the trace size of Netflix and YouTube traces, respectively). We assume that the cache is first populated only at the end of the first window since that is when the first estimate of the popularities π_k is obtained. Therefore the case of $W = 5$ million corresponds to the case where the caching is used only a few times (3 times for Netflix and once for YouYube) and therefore provides the worst performance. We observe that, as expected, the performance in terms of the average delay and remote fetched bytes per request is optimized at intermediate values of W, across all three values of the correlation parameter ρ. Further, the range over which W provides a near-optimal performance is quite wide (about 100,000 to 1,000,000 in this case). While these values seem large, note that these are aggregated over about 1400-3000 videos, so only about 33-714 requests per video on an average, which seems reasonable for obtaining good estimates of the popularity values π_k. The performance of CLoSER seems to worsen slightly with increasing correlation measure ρ. While this may seem counter-intuitive, one likely explanation is that while CLoSER relies on good estimates on the popularity distribution π_k, such stable estimates become more difficult to obtain with increasing temporal correlation. This effect may slightly offset the obvious caching benefits obtained from higher temporal correlation in video requests, resulting in slight degradation in performance with increasing temporal correlation in the video request pattern, as we observe here. However, we observe that the performance measures do not vary drastically with the change in ρ. This seems to imply that the performance of CLoSER is robust to changes in the temporal correlation of the video request pattern, and there is a large range of window length values for which the performance will be nearly optimal irrespective of the video request pattern. The results for YouTube and Netflix may not be directly comparable because of the different distributions of the video sizes (discussed in Section 5.1), which may impact the exact values of fetched bytes and playout delays observed in the two cases, in addition to the differences in their marginal distributions as well as temporal characteristics. However, the broad trends observed for these two traces are quite similar.

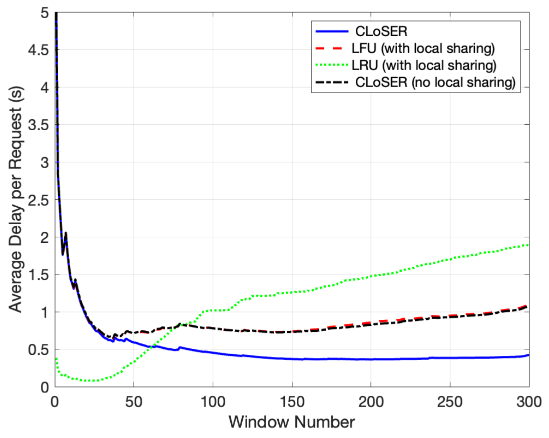

Figure 15: Comparing CLoSER with other caching methods (Youtube).

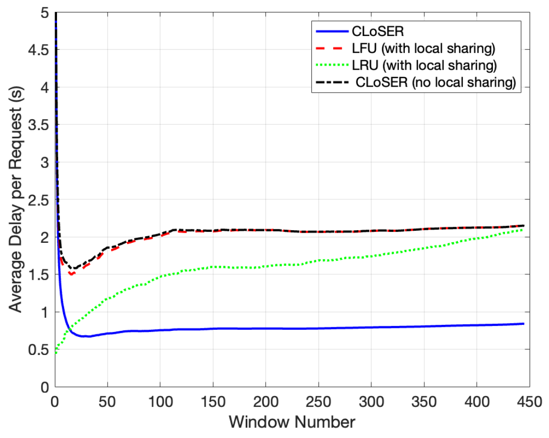

Figure 16: Comparing CLoSER with other caching methods (Netflix).

5.5. Comparison with Other Caching Methods

In this section we compare the performance of CLoSER with two other popularly used caching algorithms, Least Recently Used (LRU) and Least Frequently Used (LFU), extended to include local video sharing between the SBSs. In this extended version of LRU (LFU) that we compare CLoSER with, each cache implements LRU (LFU) individually, but preferentially obtains a requested video (that is not stored at the cache itself) from one of the other local SBS caches, before requesting from the remote server. For fairness when comparing with CLoSER which estimates the video popularity values after a window size W, we implement a windowed version of LRU and LFU. For LFU, it means that the popularity estimates are updated (in the same way as done in CLoSER) – and the cache content reoptimized according to the updated popularity values – after each window. For LRU, popularity estimates are not needed; however, the recency lists are updated after each window based on which the content is reoptimized. Also, to quantify the benefits of local sharing, we compare CLoSER with a version that does not include local sharing, i.e., Algorithm 2 is run with Phase I (Greedy Filling) alone. Tables 3, 4 and Figures 15, 16 show the comparative performance results. For CLoSER, we use $W = 30,000$ and $\alpha = 0.4$, as before.

From Tables 3 and 4, we see that compared to LRU and LFU, CLoSER saves significantly in terms of remote bandwidth. While these savings come at the cost of increased local network bandwidth use, this is what we desire – since remote bandwidth is expected to be more costly (limited) compared to local bandwidth. Figures 15 and 16 show that CLoSER provides considerable improvement in video playout delay compared to LRU and LFU, when all the algorithms have a "cold start". Furthermore, the average playout delay under CLoSER reduces quickly after start, and reaches a somewhat steady value much sooner than the other algorithms. For the YouTube trace in particular, none of the other algorithms reach a steady state by the end of the trace. This implies that the other algorithms are not able to effectively adapt to changes in

Table 3: Comparison in fetched bytes with other caching methods (YouTube).

Caching Method	**Fetched Bytes Per Request**			**Remote/Total (%)**
	Local	**Remote**	**Total**	
CLoSER	5.5 MB	2.8 MB	8.3 MB	34
LFU (with local sharing)	0.1 MB	8.3 MB	8.4 MB	99
LRU (with local sharing)	2.6 MB	16 MB	18.6 MB	86
CLoSER (no local sharing)	0	8.5 MB	8.5 MB	100

Table 4: Comparison in fetched bytes with other caching methods (Netflix).

Caching Method	**Fetched Bytes Per Request**			**Remote/Total (%)**
	Local	**Remote**	**Total**	
CLoSER	7.6 MB	1.8 MB	9.4 MB	19
LFU (with local sharing)	0.1 MB	6.3 MB	6.4 MB	98
LRU (with local sharing)	2.5 MB	5.9 MB	8.4 MB	70
CLoSER (no local sharing)	0	6.3 MB	6.3 MB	100

the video popularities as time progresses. Also note that CLoSER with no local sharing performs very close to LFU. Over time, the video popularity estimates converge to their long-term rates, and in CLoSER with no sharing (which only implements Phase I of the algorithm), all caches end up caching the same videos according to their weights $w_k = \pi_k/s_k$. In LFU as well, all the caches end up caching the same videos; although these are according to the π_k values, that does not result in a significant difference in terms of overall performance since the variation in the s_k values in our traces is not that large. While our LFU implementation supports sharing, the benefits of that are negated as all the SBSs (caches) store the same videos. So LFU ends up performing very close to CLoSER with no local sharing.

5.6. Effect of local vs remote network bandwidths

In the results shown so far, we have implicitly assumed a local network bandwidth of 100 Mbps and a remote network bandwidth of 10 Mbps. These are conservative estimates, given the emerging standards on link layer technologies. It is worth noting, however, that the performance benefits of CLoSER depend on the D/d ratios; or in other words, the ratio between the local and remote network bandwidths. While both of these are expected to grow in the near future, local network bandwidths (which is estimated to be close to 1 Tbps with 6G [49]) will likely outgrow core Internet bandwidths. This would make the D/d values larger than what we have today, implying further benefits of local sharing. To demonstrate this for the Netflix dataset (the YouTube results were similar, and omitted for brevity), in Figures 17 and 18, we plot our performance metrics against increasing D/d values by keeping D fixed at 5 secs, and decreasing d from 2.5 secs to 0.05 secs. These correspond to D/d ratios (or equivalently, the ratios of the local and remote network bandwidths) of 2 to 100. From Figure 17, we see that the remote bytes fetched per request under CLoSER steadily decreases with reducing d values (i.e., increasing D/d ratio), trading it off for increased local bytes fetched per request. The growth in these local bytes per request is however sublinear in $1/d$,

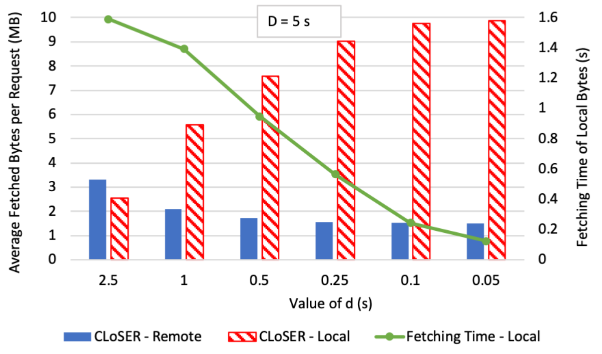

Figure 17: Fetched Bytes for different d values ($D = 5$s)

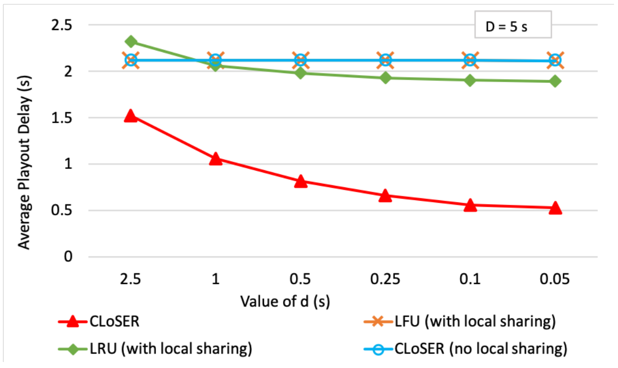

Figure 18: Playout Delay for different d values (D= 5s)

which implies that the local network bandwidth used by CLoSER *decreases relative to the local network bandwidth available* as the local network becomes faster relative to the remote network. This is shown by the curve representing the fetching time of local bytes, which decreases monotonically as D/d increases. In Figure 17, we see that CLoSER (and to some extent, LRU) is able to effectively utilize increased local network speeds to reduce the playout delay; but CLoSER performs considerably better than LRU in this aspect. As expected, with no collaboration between caches, CLoSER is not able to avail of the reduction in d values, and as D is kept constant at 5 secs, the average playout delay under CLoSER with no local sharing remains flat. LFU performs similarly to CLoSER with no local sharing, for reasons discussed earlier in Section 5.5.

5.7. Extension to multiple quality levels

In our study, we have so far assumed a fixed quality for each video, and the video size and delay numbers used in our evaluations corresponded to 1080p resolution videos for Netflix and 720p resolution videos for YouTube, as explained in Section 5.1. Current video streaming practices heavily use ABR, where each video is coded and stored at different quality levels, only one of which is chosen to be streamed to the user at any time. The choice of the quality level to stream to the user is determined based on the viewing application, user device capability, network speed etc., and may be dynamically adapted based on variation in network bandwidth [31, 32]. Since the impact of quality level fluctuations can be quite detrimental to the viewing experience [6], it is desirable that the video is cached at all the quality levels – or at least the quality levels that have a high probability of being requested. The CloSER model and algorithms can be easily modified to accommodate this extension to ABR streaming: by treating each quality level as a different video k in our system model, with its corresponding π_k, s_k, d_k and D_k values. The generalization of the "popularity ratio test" condition, as described in Section 4.3.1, extends Algorithms 1 and 2 to this ABR scenario.

Extended to the multi-quality or ABR setting, we expect the general nature of the performance results for CLoSER to remain the same as described earlier in our evaluation of non-ABR scenarios. However, caching

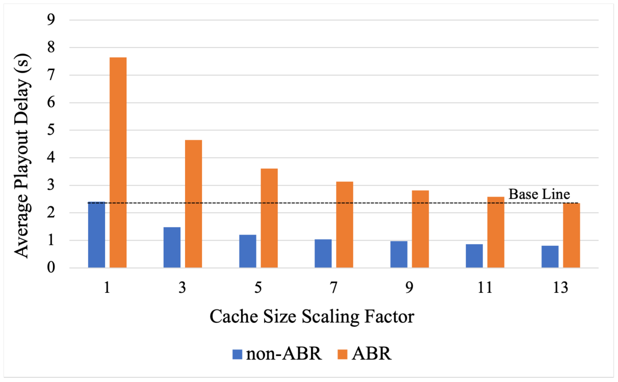

Figure 19: Playout Delay for ABR vs non-ABR (YouTube).

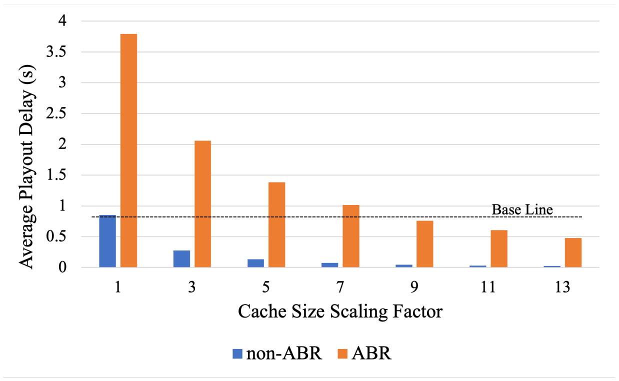

Figure 20: Playout Delay for ABR vs non-ABR (Netflix).

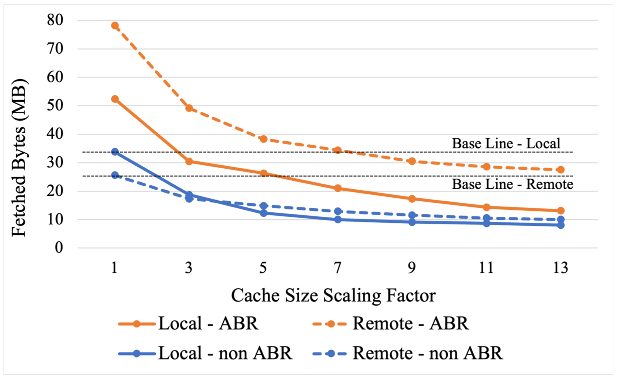

Figure 21: Fetched Bytes for ABR vs non-ABR (YouTube).

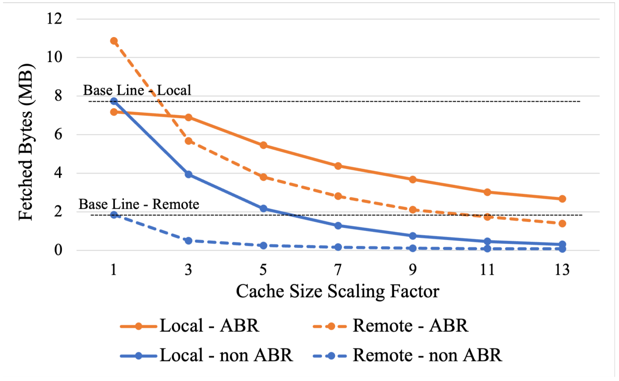

Figure 22: Fetched Bytes for ABR vs non-ABR (Netflix).

multiple (quality) representations of a video will require larger cache sizes to attain the same performance as the single-quality (non-ABR) case. We compare the ABR vs non-ABR scenarios in this section, by comparing the performance metrics (playout delay, and remote and local fetched bytes) for different values of cache sizes. The non-ABR case is simulated by assuming all videos (for both YouTube and Netflix) are requested at HD quality (1080p), while the ABR case assumes that each video is requested according to the following distribution, based on 2022 estimates [5]: SD quality (480p) for 21% of the videos, HD quality (1080p) for 57% of the videos, and UHD quality (4K) for the rest. The amount of video to be buffered are calculated as (for both YouTube and Netflix) [46, 47]: 0.5 GB per hour (8.33 MB per min) for SD quality (480p), 3 GB per hour (50 MB per min) for HD quality (1080p), and 7 GB per hour (116 MB per min) for UHD quality (4K). The rest of the parameters are consistent with those described in Section 5.1. In particular, the remote and local delays are calculated assuming 7.5 secs of video playout buffering at the above rates.

Figures 19-22 illustrate the performance of the ABR case vs the non-ABR case as the cache size for each of the 4 caches (SBSes) are increased from the default value (2GB) to multiples of it (indicated by the 'Cache Size Scaling Factor'). Thus the non-ABR case with a Cache Size Scaling Factor of 1 represents the baseline which is used in most of our prior simulations. We see that in terms of the key performance metrics, namely, the playout delay and remote fetched bytes, the ABR results are consistently worse than the non-ABR case at all cache sizes. This is expected however, and is due to two factors. The primary factor is that in the ABR case, each video must be replicated and cached at multiple quality levels; therefore attaining the same cache hit rate requires more cache space compared to the non-ABR case. The secondary factor is that the video sizes averaged over the the three quality levels (SD, HD and UHD) is slightly larger than that for HD quality, implying longer buffering time and therefore longer playout delays. It is important to note however that both YouTube and Netflix traces, the average playout delay and the remote fetched bytes for ABR case reach the baseline levels when the cache size is about 10 times that of the non-ABR case. This implies the ABR performance is expected to match the non-ABR performance when the cache space is larger, about 20GB per cache (a 10x factor), which is still quite small.

Finally, note that the YouTube results are consistently worse than Netflix in terms of absolute values. This is consistent with our earlier observations (in Sections 5.3 and 5.5), and can be explained as follows. Note that as before, about 12-24 secs of initial part of a Netflix video (uniformly distributed) at each quality level is cached, while the entire watch time of YouTube video (2.4 mins on average) is cached. The YouTube trace has fewer videos with high popularity and has a higher degree of temporal correlation in its access pattern (as explained Section 5.1) which favors improved caching perfrmance; however, it does not fully compensate for the longer size of the videos being cached. As mentioned in Section 5.4, the results for the two traces may not be directly comparable with each other due to different distributions of the video sizes being cached. It is important to note however that the results for the two traces follow the same trends. Furthermore, in both traces, a cache size scaling factor of 10x seems sufficient for the ABR case to match the performance of the non-ABR case.

6. Concluding Remarks

In this work, we considered the problem of caching and sharing of videos across a set of local small-cell base stations (SBSs), with the goal of minimizing video playout delay and reducing remote server bandwidth. Despite the integrality constraints – and non-convexity of the problem even when the integrality constraints are relaxed – we provide an algorithm (CLoSER) that yields a close-to-optimal solution and is computationally efficient. CLoSER is also implementable in a distributed manner with very limited messaging between the SBSs, and without any explicit coordination between them. Our simulations that utilized YouTube and Netflix video request traces show that an online implementation of CLoSER is able to reduce remote bandwidth usage significantly through local sharing of videos among the SBSs. Comparison with extended versions of LRU and LFU (that include local sharing) show that CLoSER not only results in significantly lower delay, but also quickly learns and adapts to changing video popularity values to maintain low average playout delays from a cold start. Our results also showed that there are some intermediate window size (cache reoptimization period) values in which the average delay and bandwidth usage results are relatively insensitive to the degree of temporal correlation in the video. This range of window sizes is still large enough such that the overall bandwidth utilized in reoptimization is small. Our simulations also established that CLoSER is able to take advantage of the increasing local-to-remote bandwidth ratio (as expected in the future) to reduce the playout delay further, unlike the other benchmark algorithms. Finally, we show how CLoSER can be extended to ABR streaming in which each video must be coded and cached at multiple qualities (bit-rates). This naturally results in reduced performance due to proliferation in the number of video versions that must be cached; nevertheless, our results show that in the ABR case the cache size needs to be scaled up adequately to match the performance of the non-ABR case.

Note that CLoSER narrowly focuses on minimizing the video playout delay, and avoids the broader question of optimizing QoE (which depends on metrics beyond playout delay [6]) over the entire duration for which the video is streamed. For short videos, however, playout delay can be the dominant factor in determining the QoE. Recent studies point out the increasing popularity of shorter videos (typically less than a minute) and increasing video browsing by users, driven by factors such as reduced attention spans of users and video marketing [2, 3, 4]. Minimizing playout latency is particularly important for such short videos. For longer videos, the streaming experience may depend more critically on other QoE metrics such as quality level fluctuations [6]. Collaborative caching over a fast local network can conceivably be used in smart ways to improve other QoE metrics as well; however, generalization of the CLoSER system model and solution approach for broader QoE optimization of longer videos remains open for future work.

The playout delays observed in the our evaluation results are generally small, mostly under 10 secs. However, it is worth noting that this is a result of the fact that we are consider 720p and 1080p quality videos for most of our study, and about 7.5 secs of video playout buffering. Compared to 1080p, these delays may increase roughly by factor of 2 for 4K video, and a factor of 4 for 8K video [47], even if the cache sizes are scaled up proportionally to accommodate the higher resolutions. We have seen this impact in Section 5.7 which considers an ABR traffic mix consisting of SD, HD and UHD quality videos. This will make the playout delays quite comparable to the actual playing time of short videos (usually less than a minute in duration, as discussed above), emphasizing further the importance of reducing the playout delay using caching as video bit-rates keep increasing in the future.

Our study on ABR ignores complex interactions between rate adaptation and caching that might exist, particularly in streaming of longer videos. For example, a user seeing a fast download of the initial part of the video (from a local cache) might be tempted to request a higher quality video which may need to be requested from the remote server thereby slowing down the streaming. Our ABR caching model assumes that users will request the initial quality based on preference, application or device capability, and is agnostic to further rate adaptation that might occur later in time, which this initial choice could possibly impact.

Furthermore, quality of the video requested by the user could also be impacted by the cache content (if made known to the user): a user may request a video at a lower quality (than what it ideally desires) if it expects a substantially shorter playout time due to the lower quality version being available in the local cache. Consideration of these factors would however require developing new models that capture user request and rate-adaptation behavior, and use of different analysis methods than those utilized in this paper. These could be interesting directions for future work.

Finally, note that in this work, we have not considered any wireless channel constraints (transmission scheduling or interference constraints) or any flexibility available in associating users to SBSs. While such considerations would make the solution partly depend on the wireless access technology, this would be a possible direction for future investigation.

7. Acknowledgment

The work of Jacob Chakareski has been supported in part by the National Science Foundation (NSF) under awards CCF-2031881, ECCS-2032387, CNS-2040088, CNS-2032033, and CNS-2106150; and by the Panasonic Chair of Sustainability at the New Jersey Institute for Technology.

References

[1] Cisco Systems. Cisco Annual Internet Report (2018–2023) White Paper. [Online]: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf (Last accessed 19 July 2023).

[2] David Bloom. YouTube Video Shorts See Giant Jump In Views In Past Year. [Online]: https://www.forbes.com/sites/dbloom/2022/08/11/youtube-video-shorts-see-giant-jump-in-views-in-past-year/ (Last accessed 19 July 2023).

[3] Chris Stokel-Walker. TikTok Wants Longer Videos—Whether You Like It or Not. [Online]: https://www.wired.co.uk/article/tiktok-wants-longer-videos-like-not (Last accessed 19 July 2023).

[4] Osama Khabab. Video Marketing Statistics: Top Trends for 2022 and Beyond. [Online]: https://www.entrepreneur.com/growing-a-business/video-marketing-statistics-top-trends-for-2022-and-beyond/425803 (Last accessed 19 July 2023).

[5] Cisco Systems. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. [Online]: https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf (Last accessed 19 July 2023).

[6] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2015.

[7] Huda S. Goian, Omar Y. Al-Jarrah, Sami Muhaidat, Yousof Al-Hammadi, Paul Yoo, and Mehrdad Dianati. Popularity-based video caching techniques for cache-enabled networks: A survey. *IEEE Access*, 7:27699–27719, 2019.

[8] Behrouz Jedari, Gopika Premsankar, Gazi Illahi, Mario Di Francesco, Abbas Mehrabi, and Antti Ylä-Jääski. Video caching, analytics, and delivery at the wireless edge: A survey and future directions. *IEEE Communications Surveys & Tutorials*, 23(1):431–471, 2021.

[9] Abdallah Khreishah and Jacob Chakareski. Collaborative caching for multicell-coordinated systems. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 257–262, 2015.

[10] Abdallah Khreishah, Jacob Chakareski, and Ammar Gharaibeh. Joint caching, routing, and channel assignment for collaborative small-cell cellular networks. *IEEE J. Selected Areas in Communications*, 34(8), August 2016.

[11] Pouya Ostovari, Jie Wu, and Abdallah Khreishah. Efficient online collaborative caching in cellular networks with multiple base stations. In *2016 IEEE 13th Intl. Conf. on Mobile Ad Hoc and Sensor Systems (MASS)*, 2016.

[12] Peng Cheng, Chuan Ma, Ming Ding, Yongjun Hu, Zihuai Lin, Yonghui Li, and Branka Vucetic. Localized small cell caching: A machine learning approach based on rating data. *IEEE Transactions on Communications*, 67(2):1663–1676, 2018.

[13] Yanting Wang, Yan Zhang, Min Sheng, and Kun Guo. On the interaction of video caching and retrieving in multi-server mobile-edge computing systems. *IEEE Wireless Communications Letters*, 8(5):1444–1447, 2019.

[14] Yanwei Liu, Jinxia Liu, Antonios Argyriou, Liming Wang, and Zhen Xu. Rendering-aware vr video caching over multi-cell mec networks. *IEEE Transactions on Vehicular Technology*, 70(3):2728–2742, 2021.

[15] Chenmeng Wang, Daquan Feng, Shengli Zhang, and Qianbin Chen. Video caching and transcoding in wireless cellular networks with mobile edge computing: A robust approach. *IEEE Transactions on Vehicular Technology*, 69(8):9234–9238, 2020.

[16] Xiaonan Liu, Nan Zhao, F. Richard Yu, Yunfei Chen, Jie Tang, and Victor C. M. Leung. Cooperative video transmission strategies via caching in small-cell networks. *IEEE Transactions on Vehicular Technology*, 67(12):12204–12217, 2018.

[17] Jun Du, Chunxiao Jiang, Erol Gelenbe, Haijun Zhang, Yong Ren, and Tony QS Quek. Double auction mechanism design for video caching in heterogeneous ultra-dense networks. *IEEE Transactions on Wireless Communications*, 18(3):1669–1683, 2019.

[18] Ahmed Alioua, Roumayssa Hamiroune, Oumayma Amiri, Manel Khelifi, Sidi mohammed Senouci, Mikael Gidlund, and Sarder Fakhrul Abedin. Incentive mechanism for competitive edge caching in 5g-enabled internet of things. *Computer Networks*, 213:109096, 2022.

[19] Chenglin Li, Laura Toni, Junni Zou, Hongkai Xiong, and Pascal Frossard. Qoe-driven mobile edge caching placement for adaptive video streaming. *IEEE Transactions on Multimedia*, 20(4):965–984, 2018.

[20] Lixing Chen, Linqi Song, Jacob Chakareski, and Jie Xu. Collaborative content placement among wireless edge caching stations with time-to-live cache. *IEEE Transactions on Multimedia*, 22(2):432–444, 2020.

[21] Manoj Kumar Somesula, Rashmi Ranjan Rout, and D.V.L.N. Somayajulu. Contact duration-aware cooperative cache placement using genetic algorithm for mobile edge networks. *Computer Networks*, 193:108062, 2021.

[22] J. Chakareski, J. Apostolopoulos, W.-T. Tan, S. Wee, and B. Girod. Distortion chains for predicting the video distortion for general packet loss patterns. In *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, volume 5, pages 1001–1004, Montreal, Canada, May 2004. IEEE.

[23] J. Chakareski, J. Apostolopoulos, S. Wee, W.-T. Tan, and B. Girod. Rate-distortion hint tracks for adaptive video streaming. *IEEE Trans. Circuits and Systems for Video Technology*, 15(10):1257–1269, October 2005. special issue on Analysis and Understanding for Video Adaptation.

[24] Osnat Mokryn, Adi Akavia, and Yossi Kanizo. Optimal cache placement with local sharing: An isp guide to the benefits of the sharing economy. *Computer Networks*, 171:107153, 2020.

[25] Dan Wu, Liang Zhou, Yueming Cai, and Yi Qian. Collaborative Caching and Matching for D2D Content Sharing. *IEEE Wireless Communications*, 25(3):43–49, 2018.

[26] S. Sinem Kafıloğlu, Gürkan Gür, and Fatih Alagöz. Cooperative Caching and Video Characteristics in D2D Edge Networks. *IEEE Communications Letters*, 24(11):2647–2651, 2020.

[27] Ming-Chun Lee and Andreas F. Molisch. Individual Preference Aware Caching Policy Design in Wireless D2D Networks. *IEEE Transactions on Wireless Communications*, 19(8):5589–5604, 2020.

[28] Haowen Xu, Rong Chen, Mingzhi Xu, Ming Jiang, and Xuming Lu. Device-to-device collaborative caching strategy based on incentive mechanism. In *2021 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 612–617, 2021.

[29] Pol Blasco and Deniz Gündüz. Learning-based optimization of cache content in a small cell base station. In *2014 IEEE International Conference on Communications (ICC)*, pages 1897–1903, 2014.

[30] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., USA, 1990.

[31] Jonathan Kua, Grenville Armitage, and Philip Branch. A survey of rate adaptation techniques for dynamic adaptive streaming over http. *Commun. Surveys Tuts.*, 19(3):1842–1866, jul 2017.

[32] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. A survey on bitrate adaptation schemes for streaming media over http. *IEEE Communications Surveys & Tutorials*, 21(1):562–585, 2019.

[33] Jacob Chakareski, Ridvan Aksu, Xavier Corbillon, Gwendal Simon, and Viswanathan Swaminathan. Viewport-driven rate-distortion optimized 360° video streaming. In *Proc. Int'l Conf. Communications*, Kansas City, MO, USA, May 2018.

[34] Cagri Ozcinar, Ana De Abreu, and Aljoscha Smolic. Viewport-aware adaptive 360° video streaming using tiles for virtual reality. *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2174–2178, 2017.

[35] Iraj Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[36] Harold P. Benson. Concave minimization: Theory, applications and algorithms. In Reiner Horst and Panos M. Pardalos, editors, *Handbook of Global Optimization*, pages 43–148. Springer US, Boston, MA, 1995.

[37] Mingzhe Chen, Walid Saad, and Changchuan Yin. Echo-liquid state deep learning for 360° content transmission and caching in wireless vr networks with cellular-connected uavs. *IEEE Transactions on Communications*, 67(9):6386–6400, 2019.

[38] Xianzhe Xu and Meixia Tao. Decentralized multi-agent multi-armed bandit learning with calibration for multi-cell caching. *IEEE Transactions on Communications*, 69(4):2457–2472, 2020.

[39] S. Krishnendu, B. N. Bharath, Vimal Bhatia, Jamel Nebhen, Michal Dobrovolny, and Tharmalingam Ratnarajah. Wireless edge caching and content popularity prediction using machine learning. *IEEE Consumer Electronics Magazine*, pages 1–1, 2022.

[40] Nikolaos Nomikos, Spyros Zoupanos, Themistoklis Charalambous, and Ioannis Krikidis. A survey on reinforcement learning-aided caching in heterogeneous mobile edge networks. *IEEE Access*, 10:4380–4413, 2022.

[41] Navneet Garg, Mathini Sellathurai, Vimal Bhatia, BN Bharath, and Tharmalingam Ratnarajah. Online content popularity prediction and learning in wireless edge caching. *IEEE Transactions on Communications*, 68(2):1087–1100, 2019.

[42] Jiajun Wu, Chenyang Yang, and Binqiang Chen. Proactive caching and bandwidth allocation in heterogenous networks by learning from historical numbers of requests. *IEEE Transactions on Communications*, 68(7):4394–4410, 2020.

[43] Jacob Chakareski. Viewport-adaptive scalable multi-user virtual reality mobile-edge streaming. *IEEE Trans. Image Processing*, 29(1):6330–6342, December 2020.

[44] Mattia Zeni, Daniele Miorandi, and Francesco De Pellegrini. Youstatanalyzer: a tool for analysing the dynamics of youtube content popularity. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, pages 286–289, 2013.

[45] James Bennett and Stan Lanning. The Netflix Prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[46] Netflix. How to control how much data Netflix uses. [Online]: https://help.netflix.com/en/node/87 (Last accessed 19 July 2023).

[47] Joe Hindy. How much data does YouTube actually use? [Online]: https://www.androidauthority.com/how-much-data-does-youtube-use-964560/ (Last accessed 19 July 2023).

[48] Mansoor Shafi, Andreas F. Molisch, Peter J. Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5G: A Tutorial Overview of Standards, Trials, Challenges, Deployment, and Practice. *IEEE Journal on Selected Areas in Communications*, 35(6):1201–1221, 2017.

[49] Christina Chaccour, Mehdi Naderi Soorki, Walid Saad, Mehdi Bennis, Petar Popovski, and Mérouane Debbah. Seven Defining Features of Terahertz (THz) Wireless Systems: A Fellowship of Communication and Sensing. *IEEE Communications Surveys & Tutorials*, 24(2):967–993, 2022.

Appendix

Proof of Theorem 1

We first note that MPEC, as stated in (1)-(3) is a convex maximization (or concave minimization) problem. Note that Phase I (Greedy Filling) maximizes the first term $d\sum_k \pi_k \sum_i x_{i,k}$. It is easy to observe that each replacement (of video k_1 by k_2 inside the while loop) in Phase II (Compare and Replace) improves the objective function in (1), which can be written as

$$\sum_k \pi_k \Bigg[d\, n_k \;+\; N(D-d)\, \min(n_k, 1) \Bigg]. \qquad (.1)$$

We now proceed to prove Theorem 1. In the following, we assume that all the π_k values are distinct. This does not result in any loss of generality: if two values $\pi_k, \pi_{k'}$ are equal, and our (pre-determined) tie-breaking rule prefers k over k', we can increase π_k by a sufficiently small amount ϵ such that pi_k and $\pi_{k'}$ become distinct, and and yet the optimum solution under the new set of π_k values remain the same as before. Note that with distinct π_k values, the optimum solution to MPEC is unique, just as it is under any given tie-breaking rule.

We divide the set $S_{>1}$ into two subsets: $S^a_{>1}$ and $S^b_{>1}$. The set $S^a_{>1}$ consists of videos in $S_{>1}$ for which no copies have been replaced by Phase II. The set $S^b_{>1}$ consists of videos in $S_{>1}$ for which at least one copy have been replaced in Phase II. Thus if n^I_k denote the number of copies of video k populated (across all SBSs) by Phase I, then $S^a_{>1} = \{k \in S_{>1} | n_k = n^I_k\}$, and $S^b_{>1} = \{k \in S_{>1} | n_k < n^I_k\}$. Note that since Phase II replaces videos from $S_{>1}$ sequentially from the largest index, $S^b_{>1}$ can can consist of most one video.

For the sake of contradiction, let us assume that CLoSER (Algorithm 1) does not compute an optimum solution, i.e., the objective function value (1) is more in OPT than in the solution computed by CLoSER. If we let n^*_k be the number of cached copies of video k in OPT, then there exists an index k' such that $n^*_{k'} > n_{k'}$. Since $\sum_k n_k = \sum_k n^*_k$ (all of the cache space is utilized by both CLoSER and OPT), there must be a $k'' \neq k'$ such that $n^*_{k''} < n_{k''}$.

We consider four cases separately, depending on whether k' belongs to $S^a_{>1}, S^b_{>1}$, S_1 or S_0.

Case A: $\underline{k' \in S^a_{>1}}$: There must be a cache i which includes k' under OPT but not under CLoSER. Since $k' \in S^a_{>1}$, it was not replaced in Phase II of CLoSER, so, all videos k in cache i' must have $\pi_k > \pi_{k'}$, else k' would have included in cache i in Phase I. Therefore, there must be a video $k'' < k$ which is included in cache i by CLoSER but not by OPT. We can then improve OPT by replacing video k' in cache i by k'', which contradicts that OPT is the optimum solution.

Case B: $\underline{k' \in S^b_{>1}}$: Again, there must be a cache i which includes k' under OPT but not under CLoSER. Then there must be a video k'' which is included in cache i under CLoSER, but not under OPT. If $k'' \in S^a_{>1}$, $\pi_{k''} > \pi_{k'}$ Then k'' must be in S_1, Since at least one copy of k' must have been replaced in Phase II, the last video in S_1, say k''' must satisfy

$$\frac{\pi_{k'''}}{\pi_{k'}} > \frac{d}{ND-(N-1)d}. \tag{.2}$$

Since $k'' \geq k'$, $\frac{\pi_{k''}}{\pi_{k'}} > \frac{d}{ND-(N-1)d}$. Then replacing k' in cache i by k'' improves OPT, since it changes the objective function in (.1) by $-\pi_{k'}d + \pi_{k''}d + N(D-d)\pi_{k''}$, which is greater than zero, from (.2). This contradicts the fact that OPT is the optimum solution.

Case C: $\underline{k' \in S_1}$: In this case, there must be multiple copies of k' in the local network under OPT, since there is exactly one copy of k' in the local network under CLoSER. Consider the k'' for which $n^*_{k''} < n_{k''}$. Consider cache i where k' is included under OPT, but not k''. If $k'' < k'$, then replacing k' by k'' in cache i changes the objective function in (.1) by at least $-\pi_{k'}d + \pi_{k''}d$, which is positive. If $k'' > k'$, then note that $n_{k''} \leq 1$; since $n^*_{k''} < n_{k''}$, $n^*_{k''}$ must be zero. Then replacing video k' by k'' in cache i changes the objective function in (.1) in OPT by $-\pi_{k'}d + \pi_{k''}d + N(D-d)\pi_{k''}$. Note that k'' is included in S_1 under CLoSER, hence must satisfy $\frac{\pi_{k''}}{\pi_{k_1}} > \frac{d}{ND-(N-1)d}$, where k_1 is the last video in $S_{>1}$. Since $\pi_{k'} < \pi_{k_1}$, it follows that $\frac{\pi_{k''}}{\pi_{k'}} > \frac{d}{ND-(N-1)d}$, which in turn implies that the change $-\pi_{k'}d + \pi_{k''}d + N(D-d)\pi_{k''}$ is greater than zero. This contradicts the fact that OPT is the optimum solution.

Case D: $\underline{k' \in S_0}$: Again, we consider the k'' for which $n^*_{k''} < n_{k''}$. Consider cache i that includes k' but not k''. Note that in this case, $k'' \in S_{>1} \cup S_1$. We first consider the case $k'' \in S_1$. In this case, since $n_{k''} = 1$, $n^*_{k''}(< n_{k''})$ must be zero. Then replacing k' by k'' in cache i changes the objective function (.1) in OPT by at least $\pi_{k''}(d + N(D-d)) - \pi_{k''}(d + N(D-d))$, which must be positive, since $k'' < k'$. Now the consider the case $k'' \in S_{>1}$. In this case, replacing k' by k'' in cache i changes the objective function (.1) in OPT by at least $-N(D-d)\pi_{k'} - \pi_{k'}d + \pi_{k''}d$. Since k' was not included under CLoSER, $\frac{\pi_{k'}}{\pi_{k_1}} < \frac{d}{ND-(N-1)d}$, for all videos $k_1 \in S_{>1}$. Therefore, we have $\frac{\pi_{k'}}{\pi_{k''}} < \frac{d}{ND-(N-1)d}$, which implies that the change in (.1) due to this replacement, $-N(D-d)\pi_{k'} - \pi_{k'}d + \pi_{k''}d$, is positive. This argues that in both cases of $k'' \in S_1$ and $k'' \in S_{>1}$, the objective function in (.1) under OPT improves by replacing k' in cache i by k'', contradicting the fact that OPT is the optimum solution.

We arrive at a contradiction in all of the four cases A, B, C, and D, thus proving that Algorithm 1 computes the optimum solution to MPEC under unit video sizes. ■

Proof of Theorem 2

The proof of the result relies on two parts. In the first part, we show that the fractional solution computed at the end of Phase II is an optimal solution to MPEC when (3) is relaxed. The essence of the argument behind this claim is similar to that in the proof of Theorem 1, and therefore we will only provide a brief outline of this part of the proof.

We then show that in Phase III, the rounding process and the dropping of fractional videos (needed to satisfy integrality constraints (3)) only adds an $O(\epsilon)$ sub-optimality gap. Note that a naive rounding down of solution computed at the end of Phase II would result in a sub-optimality gap that grows as $O(K\epsilon)$, which is undesirable as the number of videos (K) can be large.

Let us define $z_{i,k} = x_{i,k}s_k,\ \forall i, \forall k$. Then the MPEC problem in (1)-(3) can be rewritten as

$$\sum_k \frac{\pi_k}{s_k}\left[d\sum_i z_{i,k} \ + \ (D-d)\sum_i \max_{i'} z_{i',k}\right]. \tag{.3}$$

$$\text{subject to} \quad \textstyle\sum_k z_{i,k} \leq \ C_i, \quad \forall i, \quad z_{i,k} \in \ \{0, s_k\},\ \forall i, \forall k. \tag{.4}$$

Note that this representation of the general MPEC is similar to the MPEC problem formation with unit sizes, but with (i) $w_k = \frac{\pi_k}{s_k}$ replacing the popularity values π_k, (ii) $z_{i,k}$ being constrained to two values, 0 and s_k, instead of 0 and 1.

Since $z_{i,k} \in \ \{0, s_k\}$, $\max_i z_{i,k} = \min(\sum_i z_{i,k}, s_k)$, so (.3) can be equivalently written as

$$\sum_k w_k\left[d\sum_i z_{i,k} \ + \ N(D-d)\min(\sum_i z_{i,k}, s_k)\right]. \tag{.5}$$

Now letting $y_k = \sum_i z_{i,k}$, from (.5) we have that maximizing (.3) is equivalent to maximizing

$$\sum_k w_k\left[d\ y_k \ + \ N(D-d)\ \min(y_k, s_k)\right]. \tag{.6}$$

We now proceed with the first part of the proof, to show that Phases I-II maximize (.5) (or equivalently maximize (.6)) subject to (.4), but the integrality constraints being relaxed. Therefore, in this relaxed problem, whose optimal objective function value must upper bound the maximum value of (.6)) subject to (.4), the $z_{i,k}$ values can take any continuous value between 0 and s_k; and therefore, y_k can take continuous values between 0 and Ns_k. The line of reasoning behind this part of the proof is the same as that in the proof of Theorem 1, but with y_k replacing n_k, and the sets $S_{>1}, S_1$ and S_0 defined accordingly, as described in Algorithm 2. The sets $S^a_{>1}, S^b_{>1}$ are also defined similarly. Thus if y^I_k denote the aggregate size of video k populated (across all SBSs) by Phase I, then $S^a_{>1} = \{k \in S_{>1} | y_k = y^I_k\}$, and $S^b_{>1} = \{k \in S_{>1} | y_k < y^I_k\}$.

As in the proof of Theorem 1, we assume, without loss of generality, that the π_k values are all distinct. For the sake of contradiction, let us assume that CLoSER (Algorithm 2) does not compute an optimum solution, i.e., the objective function value (.5) is more in OPT than in the solution computed by CLoSER. If we let y^*_k be the aggregate size of of video k cached in OPT, then there exists an index k' such that $y^*_{k'} > y_{k'}$. Then there must be a cache i such that the amount of video k stored under OPT in the cache ($z^*_{i,k}$) is more than that stored under CLoSER ($z_{i,k}$). Since under fractional caching, no space is wasted, there must be a $k'' \neq k'$ such that $z^*_{i,k''} < z_{i,k''}$. Again, we consider the four cases separately, depending on whether k'

belongs to $S^a_{>1}, S^b_{>1}$, S_1 or S_0. For each case, the proof replaces an amount $\delta = \min(z^*_{i,k} - z_{i,k}, z_{i,k''} - z^*_{i,k''})$ of video k' in cache i with video k'', and shows that it improves the objective function value OPT, thereby arriving at a contradiction. For each case, the argument is similar to the corresponding case (A, B, C and D) in proof of Theorem 1, and therefore omitted for brevity.

Finally we analyze the rounding process in Phase III. Towards that, we analyze the change in the objective function due to the dropping of the videos during the rounding phase. Let S^i be the set of all videos included in cache i (including fractional) in the solution at the end of Phase II, and $z_{i,k}$ be the corresponding video sizes; note that $z_{i,k}$ could be fractional. Then the objective function value in (.5) at the end of Phase II, denoted by F, is expressed as

$$F = \sum_i \sum_{k \in S^i} w_k \Big[d\ z_{i,k}\ +\ (D-d)\ \min(\sum_i z_{i,k}, s_k) \Big]. \tag{.7}$$

Note that F can be written as $F = \sum_i F_i$ where $F_i = \sum_{k \in S^i} w_k \Big[d\ z_{i,k}\ +\ (D-d)\ \min(\sum_i z_{i,k}, s_k) \Big]$.

Let $\hat{S}^i \subseteq S^i$ be the set of videos in cache i after rounding (dropping) step in Phase III. Then the objective function value in (.6) at the end of Phase III, denoted by $\hat{F}$, is expressed as

$$\hat{F} = \sum_i \sum_{k \in \hat{S}^i} w_k \Big[d\ z_{i,k}\ +\ (D-d)\ \min(\sum_i z_{i,k}, s_k) \Big]. \tag{.8}$$

Note that all $z_{i,k}$ values for $k \in \hat{S}^i$ equal s_k (integral videos). If we let $\hat{F}_i = \sum_{k \in \hat{S}^i} w_k \Big[d\ z_{i,k}\ +\ (D - d)\ \min(\sum_i z_{i,k}, s_k) \Big]$, then $\hat{F} = \sum_i \hat{F}_i$.

Now to understand the difference between sets $\cup_i S^i$ and $\cup_i \hat{S}^i$, let us list the videos that can be dropped in Phase III. The videos that dropped in Phase III can be grouped into four categories:

Group A: $\underline{k' \in S^a_{>1}}$: Note that at the end of Phase I, at most one video in any cache i (the last one to be included in the cache by the greedy filling step) may be fractional. This, if still present at the end of Phase II, could be dropped at the beginning of Phase III. Thus at most one video from the set $S^a_{>1}$ could be dropped from each cache.

Group B: $\underline{k' \in S^b_{>1}}$: Since Phase II compares and replaces videos one at a time from $S_{>1}$, there is at most one video in this group, over the entire system of caches.

Group C: $\underline{k' \in S_1}$: The space reallocation for videos in S_1, carried our Phase III, will drop at most one video per cache (i.e., the video that gets truncated as the cache is being filled).

Group D: $\underline{k' \in S_0}$: Again, since the compare and replace policy considers one video in S_0 at a time, there will be at most one fractional video in S_0 in the fractional solution at the end of Phase II. Therefore, there is at most one video in this group, over the entire system of caches.

Next we claim that in any cache i, at most one video from Group A and Group C categories can get dropped, i.e, two videos - one belonging to Group A and another belonging to Group C - would not be dropped from the same cache i. To see this, note that if $k' \in S^a_{>1}$ gets dropped from cache i, none of the videos in cache i were replaced in Phase II. Therefore, a video belonging to Group C cannot be dropped from the cache. Now consider any cache i. Let us first assume that a video in Group A gets dropped from the cache in Phase III. From (.7) and (.8), we see that the difference in the objective function due to this drop, given by $\Delta F_i = F_i - \hat{F}_i$, satisfies

$$\begin{aligned}
\Delta F_i &= w_{k'} z_{i,k'} + (D-d)\, w_{k'} \min(\sum_i z_{i,k'}, s_k) \\
&\leq d w_{k'} s_{k'} + (D-d) w_{k'} s_{k'} \\
&\leq D w_{k'} s_{\max}, \qquad (.9)
\end{aligned}$$

where $s_{\max} = \max_k s_k$. Note that $\hat{S}^i$ only contains integral videos, and their weights $w_k \geq w_{k'}$ (since the lowest weight video gets dropped). Therefore,

$$\begin{aligned}
\hat{F}_i &= \sum_{k \in \hat{S}^i} w_k \Big[d\, z_{i,k} \;+\; (D-d)\; \min(\sum_i z_{i,k}, s_k) \Big] \\
&\geq w_{k'} \sum_{k \in \hat{S}^i} \Big[d\, z_{i,k} \;+\; (D-d)\; z_{i,k} \Big] \\
&\geq D w_{k'} \sum_{k \in \hat{S}^i} z_{i,k} \\
&\geq D w_{k'} (C_i - s_{\max}). \qquad (.10)
\end{aligned}$$

The last inequality in (.10) follows from the fact that the entire cache is filled up at the end of Phase II, and dropping the video k' only reduces that by $s_{\max}$. Next lets assume that the video k' that gets dropped from cache i belongs to Group C. Inequality (.12) holds in this case as well. Note that the video k' has the lowest weight among all videos in S_1 included in the cache. Since the weight of any video in S_1 is no greater than the weight of any video in $S_{>1}$, the video k' has the lowest weight among all videos in $S_{>1}$ and S_1 included in cache i. Therefore, (.10) holds in this case as well. Therefore, considering only the dropped videos in Groups A and C, from (.12) and (.10), we have $\frac{\Delta F_i}{F_i} \leq \frac{\Delta F_i}{\hat{F}_i} \leq \frac{s_{\max}}{C_i - s_{\max}}$, and

$$\begin{aligned}
\Delta F = F - \hat{F} &= \sum_i \Delta F_i \\
&\leq \sum_i \frac{s_{\max}}{C_i - s_{\max}} F_i \\
&\leq \frac{s_{\max}}{C_{\min} - s_{\max}} F, \qquad (.11)
\end{aligned}$$

where $C_{\min} = \min_i C_i$. Now let us consider the effect of the dropping the videos in Group B and Group D. Note that there is only one video in each of these groups (across all caches). For the video k' in Group B, since part of the video was replaced by a video k'' in S_1, we have $w_{k'}/w_{k''} \leq (N(D-d)+d)/d = \rho$ (say). Since k'' is the lowest weight video in S_1, it follows that $w_{k'}/w_{k'''} \leq \rho$ holds for all videos k''' included in the fractional solution in either $S_{>1}$ or S_1. For the video k' in Group D, since the video has the lowest weight among all videos included in the fractional solution, $w_{k'} \leq w_{k'''}$ holds for all videos k''' included in the fractional solution in either $S_{>1}$ or S_1. Therefore, Δf, the change in the objective function due to dropping of the videos in these two groups (B and D), can be bounded as (where $C = \sum_i C_i$):

$$\Delta f \;\leq\; \frac{(\rho+1)s_{\max}}{C - 2s_{\max}}F \;\leq\; \frac{(\rho+1)s_{\max}}{NC_{\min} - 2s_{\max}}F. \tag{.12}$$

Then, considering the drops of all the videos (a total of $N+2$ videos at most) from Groups A, B, C, D, and letting $\epsilon = s_{\max}/C_{\min}$, from (.11) and (.12), we have

$$\frac{\Delta f + \Delta F}{F} \quad\leq\quad \frac{\epsilon}{1-\epsilon} + \frac{\frac{\rho+1}{N}\epsilon}{1-\frac{2\epsilon}{N}}. \tag{.13}$$

For $N = 1$, Algorithm 2 concludes by running Phase I, followed by dropping of the last video, if fractional. Therefore, Theorem 2 obviously holds. Therefore, lets consider $N \geq 2$. Since $N \geq 2$,

$$\rho + 1 = \frac{N(D-d)+d}{d} + 1 = \frac{ND-(N-2)d}{d} \leq \frac{ND}{d}. \tag{.14}$$

From (.13), using (.14) and $N \geq 2$, $\frac{\Delta f + \Delta F}{F} \leq \frac{\epsilon}{1-\epsilon} + \frac{\frac{D}{d}\epsilon}{1-\epsilon} = \frac{(\frac{D}{d}+1)\epsilon}{1-\epsilon}$. We make the reasonable assumption that each cache is large enough to hold at least two videos. Therefore, $C_{\min} \geq 2s_{\max}$, implying $\epsilon \leq 1/2$. Then $(\frac{D}{d}+1)\epsilon/(1-\epsilon) \leq 2(\frac{D}{d}+1)\epsilon$, which is $O(\epsilon)$. The result follows. ■