Optimal Chunking and Partial Caching in Information-Centric Networks

Liang Wang, Suzan Bayhan, Jussi Kangasharju

Department of Computer Science, University of Helsinki, Finland
liang.wang@cs.helsinki.fi

***Abstract*—Caching is widely used to reduce network traffic and improve user experience. Traditionally caches store complete objects, but video files and the recent emergence of information-centric networking have highlighted a need for understanding how partial caching could be beneficial. In partial caching, objects are divided into chunks which are cached either independently or by exploiting common properties of chunks of the same file. In this paper, we identify why partial caching is beneficial, and propose a way to quantify the benefit. We develop an optimal n-Chunking algorithm with complexity $O(ns^2)$ for an s-byte file, and compare it with ϵ-optimal homogeneous chunking, where ϵ is bounded by $\mathcal{O}(n^{-2})$. Our analytical results and comparison lead to the surprising conclusion that neither sophisticated partial caching algorithm nor high complexity optimal chunking are needed in information-centric networks. Instead, simple utility-based in-network caching algorithm and low complexity homogeneous chunking are sufficient to achieve the most benefits of partial caching.**

I. Introduction

Network caching reduces network traffic by exploiting redundancy in traffic [1] and popularity of content [2]. Different caching strategies have been proposed for different cases, such as web and media caching. An important, yet not widely studied question in caching relates to whether objects should be cached in their entirety (*integral caching*) or if only parts of objects should be cached (*partial caching*). Traditionally, web caching has favored the integral approach, whereas media caching has also considered partial caching.

Information-centric networking (ICN) [3]–[6] uses caching extensively for content delivery and some ICN approaches by default divide objects into chunks, leading effectively to partial caching. This is suitable for large video files, since they are often only partially accessed [7]. Integral caching may waste cache space by storing parts of objects that are less popular than the most popular parts of the same objects. It appears intuitive to develop efficient partial caching algorithms for optimal handling of such differing popularities inside objects.

However, the reality is somewhat more nuanced, as we show in this paper. Our focus is on understanding how different ways of dividing the object (*chunking*) reflect on performance of caching and how to optimally chunk an object. We show that, while partial caching is useful for partially accessed objects, similar results can be achieved via chunking the object into enough many chunks and using simple caching algorithms. In other words, there is only a very small range of system parameters where sophisticated partial caching algorithms are needed, even with erratic object access patterns.

Even though there is a large body of literature on ICN, chunking analysis has long been overlooked. In addition, many ICN proposals [3]–[6] adopted various chunking schemes in their architecture design, but how fast the benefit from chunking vanishes is still poorly understood. Consequently, the optimal chunk size also remains as an open research question. To the best of our knowledge, there has not been any thorough theoretical analysis or empirical evaluation of the effects of different chunking schemes on caching. Especially in ICN, as content items are digitally signed and managed at chunk level, finding a good tradeoff point between caching efficiency and maintenance overhead is vital. These unsolved questions have posed a significant challenge in front of system architects, ISPs and IETF (Internet Engineering Task Force) when they define the specifications of the content management and transportation for the future Internet. We hope our work can fill this gap by providing a deep understanding on the optimal chunking and partial caching on ICN networks.

Specifically, the contributions of the paper are as follows:

- We analyze the effects of chunking and develop the concept of *popularity distribution distance* to measure the effectiveness of a chunking scheme. We derive bounds on performance of partial caching and compare the optimal chunking with naive homogeneous chunking analytically.
- We demonstrate the performance of partial caching algorithms with different chunking schemes and the experiments confirm our analysis that after a moderate number of chunks, partial caching yields no further benefits.
- Both analytical and experimental results show neither sophisticated partial caching algorithm nor optimal chunking are needed in practice. Instead, a simple utility-based caching algorithm with naive homogeneous chunking is sufficient to achieve most benefits of partial caching.

The rest of the paper is organized as follows. Section II describes our system model and Section III presents formal analysis on chunking schemes. Section IV formalizes optimal caching algorithms and evaluates their performance under different chunking schemes. Section V introduces a utility-based heuristic and compares its performance with the optimal caching algorithms. Section VI reviews related work and Section VII concludes the paper.

TABLE I: Summary of notations.

Notation	Meaning
f_i, $f_{i,j}$	File i and chunk j of file i
n_i	Number of chunks in file i
p_i, $p_{i,j}$	Popularity of file i and popularity of chunk j of file i
s_i, $s_{i,j}$	Size of file i and size of chunk j of file i
N	Number of files
M	Number of routers
L	Number of routers that are directly connected with users
R_i	Router i
C_i	Storage capacity of router i
CP	Content Provider
R_{hit}	Router that holds the requested content
$\mathcal{S}$	Set of routers between an edge router and R_{hit}
$\mathbf{X}^t$	Content distribution on routers at time t
$X_{i,j,k,k}$	Decision variable for storing $f_{i,j}$ at R_k
$X_{i,j,k,k'}$	Decision variable for retrieving $f_{i,j}$ from $R_{k'}$ by R_k
$c_{k,k'}$	Cost of retrieving one byte from $R_{k'}$ to R_k
$\mathbf{p}, \tilde{\mathbf{p}}$	Real and observed popularity vector
$\mathcal{M}, \tilde{\mathcal{M}}$	Real and observed user request patterns

II. System Model

Consider a network of M routers organized in a general topology. Let R_i denote the router i with cache storage capacity of C_i bytes. This network serves the users that generate requests for files in the set $\mathcal{I}$ with $|\mathcal{I}| = N$. We denote a file by f_i and its size by s_i. All files are stored permanently at the Content Provider (CP) which is represented as the $(M+1)^{\text{th}}$ router (R_{M+1}). Users interact only with the L *edge routers* – routers connect to the users – also referred to as leaf nodes[1]. A file f_i is divided into n_i smaller units referred to as *chunks*, and j^{th} chunk is denoted as $f_{i,j}$. Denote the probability of request for a file by this file's *popularity* p_i, and similarly denote the popularity of chunk $f_{i,j}$ by $p_{i,j}$. We refer to the popularity vector in both cases by $\mathbf{p} = [p_i]$ (or $\mathbf{p} = [p_{i,j}]$). If an edge router has the requested item in its cache, we call this a *hit* and this item is transmitted to the user directly from this router. In case of a *miss* – the case where the router does not have the item, the request is retrieved from the closest router storing this item. If the item is not stored in the network, it is retrieved from CP.

III. Analysis on Chunking

Cutting a file into smaller *chunks* improves caching performance since more fine-grained caching decisions can be made, especially when different parts of the file have different popularities. However, quantifying the effects of chunking and the resulting benefits in partial caching have largely been unexplored. We now present the relationship between chunking and performance, then quantify the benefits of partial caching, and outline the steps of an optimal chunking algorithm.

Although the common understanding is that smaller chunk size can capture user behavior (e.g., frequently-accessed parts of a video) more accurately, the smallest indivisible unit in practice is determined by many other factors, e.g., application configuration, hardware limit, packet size, and etc. For the simplicity of presentation, we refer one byte as the smallest unit for the discrete case, and a continuous real function for the continuous case to derive analytical results in closed form. Note that our choice of the word *byte* is only for distinguishing the smallest unit from chunk, rather than indicating byte-granularity chunking in realistic settings.

[1]We use node, router, and cache interchangeably.

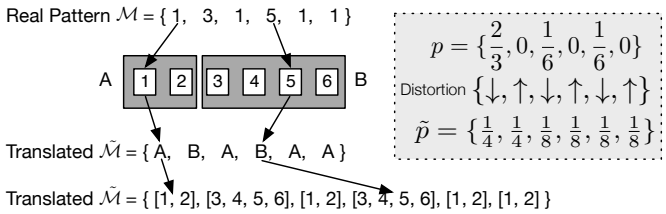


Fig. 1: Illustration of chunking effect for a chunking scheme with two chunks A and B.

A. Chunking Effect: Origin of Partial Caching Benefit

Assume that the smallest indivisible unit of a file is one byte andd the instruction booklet "P the smallest unit requested by the user is a chunk. Fig. 1 gives an example where a six-byte file is divided into two chunks A and B. Users can access individual bytes in an arbitrary manner and we denote this "real user access pattern" as $\mathcal{M}$. Because of chunking, the real access pattern has to be translated into coarser granularity chunk access pattern, from $\mathcal{M}$ to $\tilde{\mathcal{M}}$ as in Fig. 1. This distorts the popularity distribution of the bytes since unpopular bytes could be in the same chunk as popular bytes (e.g., bytes 1 and 2 in the figure), thus inflating their observed popularity. Due to this distortion, although the original popularity distribution is $\mathbf{p} = \{\frac{2}{3}, 0, \frac{1}{6}, 0, \frac{1}{6}, 0\}$, the translated popularity distribution becomes $\tilde{\mathbf{p}} = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\}$. We call this distortion from the real $\mathbf{p}$ to the translated $\tilde{\mathbf{p}}$ *chunking effect*.

Chunking effect leads to the popularity d the instruction booklet "Pof the bytes in the same chunk to be equal which often also means over- or under-estimation of popularity. Popularity estimate has direct impact on caching performance:

1) Overestimated popularity increases the chance of caching unpopular content.
2) Underestimated popularity decreases the chance of caching popular content.

Both cases lead to a failure to use cache efficiently because of wasting cache space on unpopular content.

The effect can happen anywhere in the network: at the client, at the server, or at a router whenever the chunk size is bigger than the smallest unit. Nonetheless, the effect is the same since the bytes in the same chunk will be given the same popularity and we lose the information from the real sequence. Vanichpun et al. [8] show that for most demand-driven caching algorithms (e.g., LRU, LFU), it is reasonable to assume that the closer $\tilde{\mathbf{p}}$ is to $\mathbf{p}$, the better caching decision a caching algorithm can make, therefore achieve higher caching performance.

B. Popularity Distribution Distance: Quantifying the Benefits

Multiple metrics could be used to measure information loss due to the chunking effect. However, simple difference of two distributions has very direct connection to the performance.

Vanichpun et al. [8] show that probability of an object being cached is a function of its translated probability. Let $\mathcal{C} : \mathcal{F}(\tilde{\mathcal{M}}, C) \rightarrow \mathbf{x}$ be a caching algorithm which maps a translated request pattern $\tilde{\mathcal{M}}$ and cache capacity C to a caching decision vector $\mathbf{x} = \{x_1, x_2, x_3...\}$, where x_i specifies the probability that item f_i should be kept in the cache. Let $\mathcal{I}$ denote the set of the whole content items with N elements and $\mathcal{I}_\mathcal{C}$ the items cached by $\mathcal{C}$. Assuming unit size items, the total size of the items in the cache sums to the cache capacity: $\sum_{i=1}^{N} x_i = C$. Let us introduce $\mathbf{v} = \{\frac{x_1}{C}, \ldots, \frac{x_i}{C}, \ldots, \frac{x_N}{C}\}$, which is simply normalized version of $\mathbf{x}$.

The performance of $\mathcal{C}$ can be evaluated by its (byte) hit rate $\mathcal{H}$ which is simply the joint probability of an incoming request being for a specific content f_i and this item f_i being stored in the cache. We calculate $\mathcal{H}$ as follows:

$$\mathcal{H} = P(f \in \mathcal{I}_\mathcal{C}) = \sum_{i=1}^{|\mathcal{I}|} P(f_i, f_i \in \mathcal{I}_\mathcal{C}). \tag{1}$$

Since the event that next coming request is f_i is independent from the event that f_i is in the cache, Eq. (1) can be rewritten as:

$$\mathcal{H} = \sum_{i=1}^{|\mathcal{I}|} P(f_i) P(f_i \in \mathcal{I}_\mathcal{C}) = C\mathbf{p}\mathbf{v}^T. \tag{2}$$

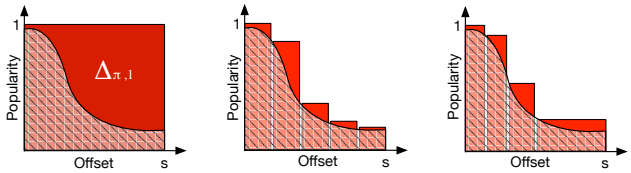
From Eq. (2), hit rate can be viewed as a function of cache size and the dot product of two distributions. Let $\mathcal{X}_k$ denote the set of k most popular objects, then the optimal caching decision is:

$$\mathbf{v}^* = \begin{cases} \frac{1}{C} & \text{if } v \in \mathcal{X}_C \\ 0 & \text{if } v \notin \mathcal{X}_C. \end{cases}$$

Caching decision $\mathbf{v}$ is completely determined by a specific caching algorithm. How close and how fast $\mathbf{v}$ converges to $\mathbf{v}^*$ is an important metric to measure the quality of a caching algorithm. Let $\mathcal{H}^*$ be the hit rate if the caching decision is optimal ($\mathbf{v}^*$). Then, the performance gap between a non-optimal and the optimal scheme is calculated as follows:

$$\Delta\mathcal{H} = \mathcal{H}^* - \tilde{\mathcal{H}} = C\mathbf{p}(\mathbf{v}^* - \mathbf{v})^T. \tag{3}$$

Similarly, Eq. (3) can also be used to compare the performance difference of two caching algorithms or two chunking schemes under a specific caching algorithm. However, in most cases, $\mathcal{C} : \mathcal{F}(\tilde{\mathcal{M}}, C) \rightarrow \mathbf{v}$ is too complicated to provide any useful information. To get around this, we measure the "expected performance loss" instead. As argued above, the less information we lose in the request series, the better caching decision we can make. Then, given the distribution $\tilde{\mathbf{p}}$ which is expected by a caching algorithm, what would be the performance loss if the actual content popularity based on user request is $\mathbf{p}$? Expected performance loss can be calculated as follows:



(a) Integral caching. (b) Homogeneous chunks. (c) Heterogeneous chunks.

Fig. 2: Benefit of chunking.

$$\Delta\mathcal{H} = \mathcal{H} - \tilde{\mathcal{H}} = C(\mathbf{p} - \tilde{\mathbf{p}})\mathbf{v}^T \propto (\mathbf{p} - \tilde{\mathbf{p}}). \tag{4}$$

In a sense, the "expected loss" shows the potential benefit from partial caching. If we define $|\mathbf{p} - \tilde{\mathbf{p}}|$ as the *popularity distribution distance* to measure the difference of two distributions, Eq. (4) shows that the potential performance gain of a partial caching algorithm positively correlates to the popularity distribution distance.

C. Performance Bound: Decaying Speed of the Benefit

Let $\pi_i(y)$ denote the probability of accessing the byte at position y of file f_i. In other words, it is the popularity of y^{th} byte in f_i. We assume that $\pi_i(y)$ is a continuous function and has at least second derivative over $[0, s_i]$. We further denote $\Pi_i(y)$ as the popularity observed from a specific chunking. Since all bytes in a chunk have the same popularity, $\Pi_i(y)$ is a step function. While $\pi_i(y)$ is the real popularity distribution (as the discrete $\mathbf{p}$), $\Pi_i(y)$ is the perceived one as $\tilde{\mathbf{p}}$.

Fig. 2 illustrates why we can benefit from chunking. x-axis is the offset in a file, and y-axis is the conditional probability of this offset byte being accessed, i.e., the popularity of the specific byte located at this offset. The curved boundary of the shaded area in the figure represents the real popularity $\pi_i(y)$. The solid red bars represent the popularity assigned for each chunk in the file. *Integral caching* can be considered as a special case of partial caching with only one chunk and $\Pi_i(y) = 1$ as Fig. 2a shows. The area above the curve $\pi_i(y)$ is the overestimation part. Naturally, chunking a file into smaller pieces reduces the difference of the two areas.

From Eq. (4), we can interpret that quantifying the benefit of chunking is equivalent to measuring the distance of two models derived from $\pi_i(y)$ and $\Pi_i(y)$ respectively. We calculate the chunking benefit as follows:

$$\Delta_{\pi_i,\Pi_i} = \frac{1}{s_i} \int_0^{s_i} \big|\pi_i(y) - \Pi_i(y)\big| dy. \tag{5}$$

Δ_{π_i,Π_i} is normalized over the file size so that we can compare files of different sizes. Based on the definition of Δ_{π_i,Π_i}, we can further define the upper bound of benefit from partial caching to integral caching as $\Delta_{\pi_i,1}$, where 1 stands for uniform chunk popularity function $\Pi_i(y) = 1$ for all $y \leqslant s_i$. Obviously, the smaller the Δ_{π_i,Π_i} is, the more accurately $\Pi_i(y)$ describes user's real access pattern. Then, improving chunking performance boils down to chunking a file in a way

that minimizes the difference Δ_{π_i,Π_i}. A follow-up question is how fast it converges to the actual access pattern, i.e., how quickly Δ_{π_i,Π_i} converges to 0.

Because $\pi_i(y)$ may be arbitrary distribution, deriving the exact decaying speed is difficult. However, $\Pi_i(y)$ is a step function, and from Eq. (5), we can see the integration on the second term is actually the Riemann sum of $\pi_i(y)$ over $[0, s_i]$, as the area of rectangles shows in Fig. 2. Suppose we use midpoint Riemann sum for obtaining a tighter bound, and let the middle point represented by $\bar{y}_k = \frac{1}{2}(y_{k-1} + y_k)$. Then $\Pi_i(y)$ can be defined as follows:

$$\Pi_i(y) = \pi_i(\frac{y_{k-1} + y_k}{2}) = \pi_i(\bar{y}_k) \quad \forall y \in [y_{k-1}, y_k).$$

Using Taylor series expansion on $\pi_i(y)$, we calculate the bound of the popularity distribution distance for the k^{th} chunk as follows:

$$\begin{aligned}\left|\delta_{i,k}\right| &= \left|\int_{y_{k-1}}^{y_k} \left|\pi_i(y) - \Pi_i(y)\right| dy\right| \\ &= \int_{y_{k-1}}^{y_k} \left|\pi_i'(\bar{y}_k)(y - \bar{y}_k) + \frac{1}{2}\pi_i''(\bar{y}_k)(y - \bar{y}_k)^2\right| dy \\ &\leq \frac{1}{24}(y_k - y_{k-1})^3 \max_{[0,s_i]} |\pi_i''(y)|.\end{aligned}$$

Summing over n chunks, we can get the bound for overall popularity distribution distance as follows:

$$\begin{aligned}\left|\Delta_{\pi_i,\Pi_i}\right| &= \left|\frac{1}{s_i}\int_0^{s_i} \left|\pi_i(y) - \Pi_i(y)\right| dy\right| \\ &\leq \frac{1}{s_i} \times [\frac{1}{24} \times \frac{(s_i - 0)^3}{n^2} \times \max_{[0,s_i]} |\pi_i''(y)|] \\ &= \frac{\max_{[0,s_i]} |\pi_i''(y)|}{24} \times \frac{s_i^2}{n^2}.\end{aligned}$$

The second term is the square of the average chunk size. If we consider the chunk popularity and file size as given, we can replace the first term with constant c. s_i/n is the average chunk size, denoted as $\bar{s}$. The formula[2] can be rewritten as:

$$\left|\Delta_{\pi_i,\Pi_i}\right| \leq c \times (\frac{s_i}{n})^2 = \mathcal{O}(\bar{s}^2) = \mathcal{O}(n^{-2}). \tag{6}$$

This shows that the popularity distribution distance is bounded by the square of the average chunk size. For a given file, the convergence speed of model distance is inversely proportional to n^2. In practical terms, this implies that there is a specific number of chunks for every file after which additional chunks bring only negligible benefits in performance. In Section III-D we will discuss the overheads caused by optimal chunking, and in Section III-E we will show the validity of Eq.(6) via experiments.

Conclusion:

1) *Performance of partial caching strategy positively correlates to the popularity distribution distance, which decays with the speed bounded by $\mathcal{O}(\bar{s}^2)$ and $\mathcal{O}(n^{-2})$.*
2) *Given a target performance, the number of chunks needed increases linearly as file size grows.*

[2]If Riemann sum is calculated using alternate methods, e.g., left, right, minimum or maximum sum, the bound changes to $\mathcal{O}(\bar{s})$ and $\mathcal{O}(n^{-1})$.

D. Optimal Chunking and Complexity Analysis

Despite the fact that smaller chunk size helps in improving the caching performance, we should avoid too many chunks due to the maintenance overheads and other practical considerations (e.g, hardware limitations, minimum video clip size, and protocol overhead). In this section, we present an optimal chunking algorithm and analyze its complexity. First, for a given number of chunks n, we outline how to divide a file into chunks to achieve the minimum Δ_{π_i,Π_i}. Next, we find the smallest number of chunks to attain a target Δ_{π_i,Π_i}.

The following discussion revolves around these two questions and details the algorithmic analysis. In practice, the first question focuses on locating the optimal dividing points within a file to achieve the best caching performance, whereas the second question focuses on finding out the least number of chunks needed for optimal chunking. The two questions together give an algorithmic solution to obtain the optimal chunking scheme in an ICN system. However, by taking the actual computation complexity into account, our following analysis also attests that such optimal solution is nothing but a white elephant function in ICN architectures.

Question 1: Given that an s-byte file is to be divided into n chunks where $s \gg n$, how can we find the optimal chunking that achieves the minimum popularity distribution distance $\Delta_{\min}$? Assuming that the popularity of each byte is known and a file can be partitioned into chunks from any point, a naive algorithm checks all possible partitions and selects the one achieving $\Delta_{\min}$. Since there are $\binom{s-1}{n-1}$ ways of dividing an s-byte file into n chunks, the computational complexity equals to $\binom{s-1}{n-1} = \frac{(s-1)!}{(n-1)!(s-n)!} = \frac{1}{(n-1)!}s^{n-1} + o(s^{n-1})$ showing the time complexity of naive algorithm to be $\mathcal{O}(s^{n-1})$, and space complexity to be $\mathcal{O}(n)$.

This optimization problem exhibits obvious "optimal substructure". Assume we look for a solution which produces $n = n_1 + n_2$ chunks, and the first n_1 chunks cover $[0, i]$ bytes, and the remaining n_2 chunks cover $[i + 1, s - 1]$ bytes. We can first focus on $[0, i]$ and optimize it, then turn to optimize $[i + 1, s - 1]$. By combining the optimal solutions of both parts, we have the optimal partition of the whole file. Let $\Delta|_{[a,b]}^n$ be the minimum popularity distribution distance for the file's bytes between $[a, b]$ in case of n chunks. The minimum distance can be recursively found as follows:

$$\Delta_{\min} = \Delta|_{[0,s-1]}^n = \min_{0 \leq i < s} (\Delta|_{[0,i]}^1 + \Delta|_{[i+1,s-1]}^{n-1}). \tag{7}$$

The top-down recursive solution Eq. (7) also tries all the possible cutting points. The induced recursive tree has degree $\mathcal{O}(s)$, and depth $n-1$ (restricted by the number of chunks n). Although simple analysis shows that it suffers from the same time complexity $\mathcal{O}(s^{n-1})$ as the naive algorithm, its optimal substructure property helps us reducing the search complexity by eliminating redundant calculations.

Algorithm 1 shows our heterogeneous n-Chunking algorithm using bottom-up dynamic programming. The algorithm first constructs two tables X and X' of size $n \times s$. Element $X[i, j]$ stores the optimal Δ over bytes $[0, j]$ by cutting it

Algorithm 1 Heterogeneous n-Chunking

1: **Input:** s-byte file, n chunks
2: **Output:** chunking scheme
3: Construct three tables X, X' and Y.
4: Set $X[i,j] = +\infty \quad \forall i \in [1, n-1], \forall j \in [1, s]$
5: Set $X[i,0] = \Delta|_{[0,0]}^{i+1} \quad \forall i \in [0, n-1]$
6: Set $X[0,j] = \Delta|_{[0,j]}^{1} \quad \forall j \in [0, s-1]$
7: Set $Y[i,j] = \Delta|_{[i,j]}^{1} \quad \forall i \leq j \in [0, s-1]$
8: **for** $i = 1 \rightarrow n-1$ **do**
9: **for** $j = 1 \rightarrow s-1$ **do**
10: **for** $j' = 1 \rightarrow j-1$ **do**
11: **if** $X[i,j] > X[i-1,j'] + Y[j'+1,j]$ **then**
12: $X[i,j] = X[i-1,j'] + Y[j'+1,j]$
13: $X'[i,j] = j'$
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: Return X, X'

into $i+1$ chunks. $X'[i,j]$ stores the starting point of the last chunk in an optimal chunking scheme over $[0,j]$ with $i+1$ chunks. Then the algorithm constructs another table Y of size s^2, where $Y[i,j]$ stores $\Delta|_{[i,j]}^{1}$. Filling $X[0,j]$, $X[i,0]$, and $Y[i,j]$ are trivial. Lines (8 – 17) are for the general case; the three loops build the table row by row by filling each cell $X[i,j]$ with optimal $\Delta|_{[0,j]}^{i+1}$ with the given chunk number (i.e. $i+1$) and data range (i.e. $[0,j]$). There are $\mathcal{O}(n \times s)$ entries in X, and for each entry, we need to consider $\mathcal{O}(s)$ cases. Therefore the time complexity of n-Chunking is $\mathcal{O}(ns^2)$, and space complexity is $\mathcal{O}(s^2)$. The optimal chunking scheme can be easily constructed from X' returned by n-Chunking and the corresponding $\Delta_{\min}$ is stored at $X[n-1, s-1]$.

Question 2: Given a target popularity distribution distance Δ, what is the minimum number of chunks $n_{\min}$ that achieves the desired Δ? This problem can be considered as an extension of the previous one, and can be solved by slightly adjusting n-Chunking algorithm. We can replace the first deterministic n-round loop with a *while* loop, where we keep comparing Δ with $X[n', s-1]$ (n' is the current round), and break the loop when $\Delta > X[n', s-1]$. Then $n_{\min} = n'$ is the minimum number of chunks to achieve Δ, and X' contains the corresponding chunking scheme. Tables X and X' grow during the calculation. The complexity then depends on the number of iterations $n_{\min}$, and our analysis in Section III-C shows $n_{\min} \leq c^{\frac{1}{2}} s \left|\Delta\right|^{-\frac{1}{2}}$. $\mathcal{O}(n_{\min} s^2)$ and $\mathcal{O}(s^2)$ are time and space complexities for solving Question 2, respectively.

E. Homogeneous Chunking: Low-Complexity Alternative

Figure 2b and 2c show examples of homogeneous and heterogeneous chunking schemes. n-Chunking is an example of heterogeneous chunking and it requires file internal popularity distribution; its complexity is also considerably high. On the contrary, a homogeneous chunking does not require any information about popularity, as it simply divides the object into equal-sized chunks. It can be easily extrapolated from the conclusions in Section III-C, the performance difference of n-Chunking and a homogeneous chunking is bounded by $\mathcal{O}(\bar{s}^2)$ and $\mathcal{O}(n^{-2})$. In other words, as the number of chunks increases, the difference between the schemes diminishes.

Even though there are concerns that the increased number of chunks may bring up management overheads, [9], [10] actually showed the feasibility of caching many small chunks without degrading the performance in reality. Practically, it indicates there is no need to incorporate complicated optimal chunking scheme in the system. As we show in Section IV, after a moderate number of chunks, there is essentially no difference between the optimal and the homogeneous chunking.

Conclusion: *Homogeneous chunking is ϵ-optimal where $\epsilon = \frac{cs_i^2}{n^2}$. In other words, the difference between homogeneous chunking and n-Chunking is bounded by $\mathcal{O}(n^{-2})$.*

IV. Analysis on Caching Systems

To verify our analysis in Section III and obtain performance bounds, we now evaluate different chunking schemes on a caching system. Please refer to Table I for the notation.

A. Caching System Model

Assume that there is a centralized entity aware of the network conditions. In particular, the content distribution at time t denoted by $\mathbf{X}^t = [x_{i,j,k}^t]$ is available to this entity. If chunk $f_{i,j}$ is stored at node R_k, then $x_{i,j,k}^t$ is 1 and zero otherwise. We devise a strategy called *dynamic partial caching* (P_{DyPa}) that gives a decision at each time instant when a user requests a chunk $f_{u,v}$, i.e., it dynamically adapts the cached contents. Suppose at time t chunk $f_{u,v}$ is stored at node R_{hit} and a user requests via a leaf node R_l. R_l will retrieve this item from R_{hit}, and each router on the path from R_{hit} to R_l must decide: cache this object or not. In case the item is cached in any of the nodes, some items stored in the cache may need to be evicted from the cache. We refer the set of all these intermediate nodes as $\mathcal{S}$.

An optimal caching strategy minimizes the cost of serving the whole user requests by storing the items at the most appropriate routers and by favoring the most popular chunks of the most popular files. Let $c_{k,k'}$ denote the cost function for fetching one byte from $R_{k'}$ to R_k. It reflects the distance between the two entities which can be calculated using shortest path algorithms, e.g., Dijkstra. The optimal strategy decides if chunk $f_{i,j}$ is to be stored at R_k and if not, from which router $R_{k'}$ to be fetched. Since this decision determines the new content distribution at time $t+1$, we define our decision variables as $\mathbf{X}^{t+1}$. Let our binary decision variable $x_{i,j,k}^{t+1}$ be 1 if chunk $f_{i,j}$ is to be stored at R_k. Similarly, let $x_{i,j,k,k'}^{t+1}$ be 1 if R_k downloads $f_{i,j}$ from $R_{k'}$. For harmony of notation, we re-define the content distribution by $\mathbf{X}^t = [x_{i,j,k,k}^t]$. Given $\mathbf{X}^t$, the number of chunks in a file i (n_i), file popularity (p_i), chunk popularity ($p_{i,j}$), and chunk size ($s_{i,j}$) information, using the

approach provided in [11], we can formulate P_{DyPa} as follows:

$$P_{\mathrm{DyPa}}: \min\Big(\sum_{k=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{n_i}\sum_{k'=1}^{M+1} s_{i,j}p_i p_{i,j} c_{k,k'} x^{t+1}_{i,j,k,k'} x^{t+1}_{i,j,k',k'} + s_{u,v}p_u p_{u,v}\Big(\sum_{k=1}^{L}\sum_{\forall R_{k'}\in\mathcal{S}\cup R_{M+1}} c_{k,k'} x^{t+1}_{u,v,k,k'}\Big)\Big) \quad (8)$$

subject to:

$$\sum_{i=1}^{N}\sum_{j=1}^{n_i} s_{i,j}x^{t}_{i,j,k,k}x^{t+1}_{i,j,k,k} + s_{u,v}x^{t+1}_{u,v,k,k}(1-x^{t}_{u,v,k,k}) \le C_k, \forall R_k \quad (9)$$

$$x^{t+1}_{i,j,k,k'} \le x^{t+1}_{i,j,k',k'} \qquad \forall i,\forall j,\forall k,\forall k' \quad (10)$$

$$1 \le \sum_{k'=1}^{M+1} x^{t+1}_{i,j,k,k'} \quad \forall i,\forall j,\forall k \quad (11)$$

$$x^{t+1}_{i,j,M+1,M+1} = 1 \quad \forall i,\forall j \quad (12)$$

$$x^{t+1}_{i,j,k,k'} \in \{0,1\} \qquad \forall i,\forall j,\forall k,\forall k'. \quad (13)$$

Our objective (8) includes the cost of serving the request from the edge router's cache as well as the cost of serving from any other content router in the ISP network and CP (i.e., R_{M+1}). The first term in (8) represents the cost of serving the requests for the contents already existing in the system whereas the second term is for serving the current request for item $f_{u,v}$. Please note that if $x_{i,j,k,k} = 1$, then $f_{i,j}$ is stored in R_k. Const. (9) ensures the total size of items to be stored in a cache cannot exceed cache capacity. Const. (10) reflects the fact that $f_{i,j}$ can be fetched from $R_{k'}$ only if $R_{k'}$ stores $f_{i,j}$. Const. (11) forces the content item to be served from some location (i.e., local cache, another router's cache, or the CP) while Const. (12) states that all items are permanently stored in the CP. Const. (13) defines the type of variables. The problem in (8-13) is an integer linear programming problem (ILP) which can be solved by an optimization software once $c_{k,k'}$ values are computed.

If we assume that file and chunk popularities already contain sufficient information to describe user behavior, we can further simplify the optimal caching problem into a static content placement problem, i.e., the set of items to be stored at each cache is decided once and no changes are done. In this approach, content distribution is time-invariant leading to $\mathbf{X}^{t+1} = \mathbf{X}^t$ for all t. Hence, we can simplify P_{DyPa} using this equality to derive P_{StPa} as follows:

$$P_{\mathrm{StPa}}: \min \sum_{k=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{n_i}\sum_{k'=1}^{M+1} s_{i,j}p_i p_{i,j} c_{k,k'} x_{i,j,k,k'}. \quad (14)$$

The constraints would need to be similarly modified. Since integral caching is a special case of partial caching with $n_i = 1$ in Eq.(14), we skip the formulation for dynamic integral caching (DyIn).

Compared with the dynamic model, the static model does not take current content distribution into account and is unaware of the specific user request sequence. The static model aims to achieve the long-term optimality in content placement while the dynamic model attempts to find the best next caching decision based on the current content distribution in the network. Besides both problems being hard to solve as ILP problems [12], they require centralized global knowledge which is infeasible in real networks. We use these models as benchmarks for our lower-complexity heuristic in Section V.

B. Metrics and Setup

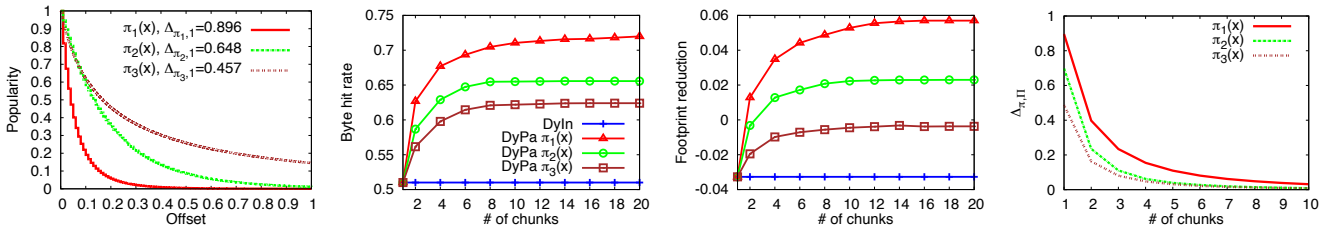
We use the following metrics to assess the performance of the presented caching/chunking schemes:

- **Byte hit rate (BHR):** Byte hit rate is the percent of requested data that can be served by the cache within the network. It measures savings in outgoing traffic.
- **Footprint reduction (FPR):** Footprint reduction is the product of traffic volume and the distance it travels in the network. It measures traffic reduction inside the network compared to retrieving the content from the CP.
- **Cache-level similarity** (γ_c) measures how much the same content is stored at the caches of two caching strategies subject to analysis. Hence, cache-level similarity measures the average similarity over all routers in the network under two caching strategies. We use *Jaccard similarity* [13] for evaluating the similarity of two routers.
- **Network-level similarity** (γ_n) considers the whole caches as a single huge cache and compares the similarity of the caches of the two networks under comparison. Network-level similarity calculated using Jaccard similarity ignores the exact storage location and focuses on if an item is stored in the network or not.

Cheng et al. [14] show that Youtube videos' popularity follows Weibull distribution with shape parameter $k = 0.513$; we use this setting in the evaluation. We also use Weibull distribution to model chunk popularity with different shape parameters. Because there is no evidence showing that file size correlates with its popularity, we select file sizes from $(5, 15)$ MB uniformly. Each video is chunked with the heterogeneous n-Chunking algorithm and user request pattern follows Independent Reference Model (IRM). We evaluated our model on both realistic ISP networks and synthetic ones with regular tree structure. In the following, we present the results of regular tree topologies as we observe very similar results for both settings. Moreover, as discussed in [15] effective topology essentially boils down to one or multiple distribution trees. Therefore, for the purpose of simplicity and also due to the space limitations, we only present the results on the 4-level binary-tree topology with model parameters set to $M = 5$, $N = 4000$ and $C = 2$ GB. We verified various parameter settings on different topologies and results agree with those shown here.

C. Partial Caching vs. Integral Caching

Fig. 3a shows three Weibull distributions with different shape parameters (0.4, 0.6 and 0.8 for π_1, π_2 and π_3 respectively) to model the internal popularity. The partial caching benefit for these distributions are $\Delta_{\pi_1,1} = 0.896$,



(a) Internal popularity as a function of offset.

(b) BHR as a function of # of chunks.

(c) FPR as a function of # of chunks.

(d) Popularity distance as a function of # of chunks.

Fig. 3: Effect of chunk popularity on BHR and FPR in dynamic model.

$\Delta_{\pi_2,1} = 0.648$ and $\Delta_{\pi_3,1} = 0.457$. The calculation implies distribution π_1 has the largest benefit and π_3 has the smallest. Results in Fig. 3b and 3c match the analysis quite well, e.g., Fig. 3b shows that BHR increases by 21% for π_1 and 11% for π_3, respectively. Eq. (4) suggests given the same cache model and caching algorithm, the improvement ratio for these distributions should equal to their partial caching benefit ratio. The result shows these two values are indeed very close to each other ($\Delta_{\pi_1,1} : \Delta_{\pi_2,1} : \Delta_{\pi_3,1} \approx 2 : 1.4 : 1$). Compared with integral caching, chunking also helps FPR improve from -4% to 6%. The reason for a negative FPR is that the model searches for content in the whole network which may lead to fetching content from somewhere further than CP. The results show that the closer the chunk popularity is to a uniform distribution (π_3), the less benefits partial caching brings.

Although $\Delta_{\pi,1}$ can accurately predict the potential benefit, it does not measure how fast it vanishes. Fig. 3d plots the popularity distance Δ_{π_i,Π_i} as a function of number of chunks and shows that a few chunks can significantly reduce popularity distance Δ_{π_i,Π_i}. This is also visible in Figs. 3b and 3c which show that the metric in question improves fastest when the number of chunks is small. Adding chunks beyond 10 yields negligible additional benefits. The slower the convergence (π_1), the larger the benefits from partial caching.

D. n-Chunking vs. Homogeneous Chunking

As mentioned in Section III-E, the differences between chunking schemes diminish as the number of chunks increases. Fig. 4a shows the performance of our n-Chunking scheme and a homogeneous chunking scheme, and illustrate that the difference disappears when there are 20 chunks. Fig. 4b shows how BHR changes as a function of number of chunks for different file sizes. The three black circles on each line mark the number of chunks needed by n-Chunking algorithm to achieve $\Delta_{\pi,\Pi} = 0.1$. Note that this number increases linearly as file size grows, i.e., $\Delta_{\pi,\Pi} = 0.1$ is achieved at exactly same chunk size (0.8 MB). It verifies our analysis in Section III-C, showing that average chunk size is a key factor of performance gain. The results also indicate that the speed at which partial caching benefit decays will decrease as file grows because we need more chunks to achieve the same gain. Increasing file size also increases the number of chunks and even though they follow the same popularity distribution, the tail becomes heavier and degrades caching performance.

We also compared the number of chunks a scheme will generate to achieve the target $\Delta_{\pi,\Pi}$. In the experiment, we use Weibull distribution with different parameters to model the internal popularity of a 10 MB file ($\lambda \in [1, 5]$, $k \in [0.5, 5]$). Fig. 4c shows that for a small $\Delta_{\pi,\Pi}$, n-Chunking always uses less chunks, with about 40%–50% improvement compared with homogeneous chunking. However, as $\Delta_{\pi,\Pi}$ increases, two schemes eventually become the same since one or two chunks are sufficient to achieve the goal.

In Section IV-A we defined both a dynamic (DyPa) and a static (StPa) caching strategy. Intuitively, DyPa should be more effective and our results confirm this. For space reasons, we present only the cache- and network-level similarities between these two approaches. Fig. 4d measures what ratio of the content in DyPa is the same as that in StPa in each step. Initially the similarity is low, but as the simulation runs longer, network-level similarity increases, indicating that they store similar content. A partial explanation is that we do not consider changes to content popularity, thus this result is to be expected. However, the cache-level similarity remains very low indicating that content placement is very different in the two cases. Fig. 4d also shows that n-Chunking increases the similarity at both network and cache level.

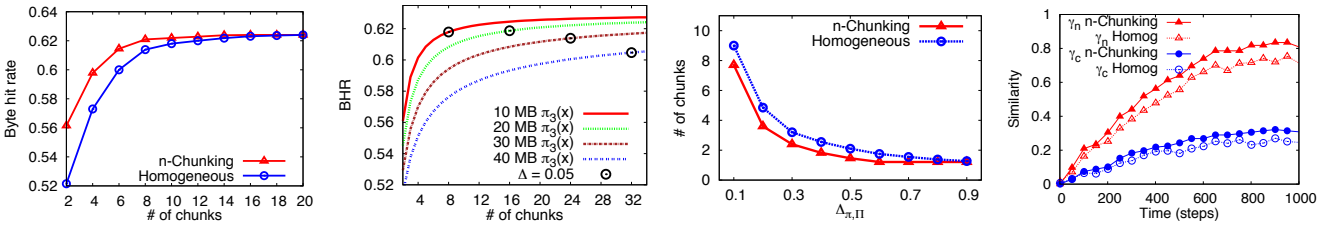
V. Low-Complexity Partial Caching Solution

We now propose a low-complexity heuristic caching strategy combined with naive homogeneous chunking at the server and evaluate its performance by comparing against the optimal solution on tree topologies. We then compare against other solutions in the literature on the realistic ISP topologies. The results are from the emulation experiments on our testbed.

A. HECTIC: HighEst CosT Item Caching

HECTIC is a low-complexity caching strategy devised to achieve the objectives in Section IV-A. To illustrate the design rationale of HECTIC, we consider three components – *admission policy*, *eviction policy* and *cooperation policy*.

Admission policy determines which objects are to be cached and which not. But in a network of caches, caches far from clients only receive a filtered version of the actual user request pattern because of the hits in the caches closer to the client. This so-called *filtering effect* [16], [17] may significantly impact the effectiveness of a hierarchical caching network. In [18] we proposed Cachedbit which spreads content



(a) BHR as a function of # of chunks.

(b) BHR under different average file size.

(c) # of chunks as a function of Δ.

(d) Similarity as a function of # of timesteps.

Fig. 4: Comparison of n-Chunking and homogeneous chunking. Similarity metrics between dynamic and static caching.

probabilistically in the caches along a path. The caching probability at R_k is the reciprocal of its distance from the client: $p_k = \frac{1}{d_{c,R_k}}$. As we showed in [18], Cachedbit is easy to implement, has low overheads, and provides some immunity against filtering effect.

For eviction policy, we propose a utility-based replacement algorithm. Let *chunk utility* of chunk $f_{i,j}$ at router R_k be:

$$u^k_{i,j} = s_{i,j} p_i p_{i,j} c_{k,M+1} \quad \forall f_{i,j} \in C_k \tag{15}$$

where $c_{k,M+1}$ represents the cost of retrieving this chunk from CP. Utility function (15) considers chunk size, its popularity, and the efforts of retrieving it. A cache will evict the chunk with the smallest utility first.

Cooperation between caches can reduce duplicate copies and allow for more efficient use of storage, but at the cost of communication overheads. More specifically, we can consider HECTIC's cost from two perspectives: node collaboration and chunking. The collaboration cost depends on the collaborative algorithm design and has been well analyzed in [19], [20]. HECTIC is en-route strategy with very limited collaboration range, therefore introduces negligible overhead. From chunking perspective, overly small chunk should be avoided due to the transmission cost. However, as our chunking analysis in Section III shows the benefit from small chunks decays quickly. Furthermore, the realistic evaluation in Section V-C attests that for all Youtube videos, more than 8 chunks per file (i.e. smaller than 0.8-1 MB per chunk) will not bring extra performance gain. Obviously, the nearly-optimal chunk size in most context is much larger than the actual chunk size used in many ICN proposals [9], [10], [21].

As a simple solution for reducing overhead, we use one bit in the packet to indicate if an object is cached in an upstream cache [18]. Note that the actual technical modification varies in different ICN proposals, and such modification may not be trivial in certain proposals if both header and content are cryptographically bounded. Take CCNx as an example: according to the newest specification [22], there is a 16-bit *reserved* field in the header which can be used to provide additional information [3]. Obviously, the communication protocol needs to be adapted consistently to understand the semantic meaning of the extension. This ensures that on a path from the CP to one client, at most one copy of a chunk exists, but paths to other clients may contain additional copies; this improves both BHR and FPR.

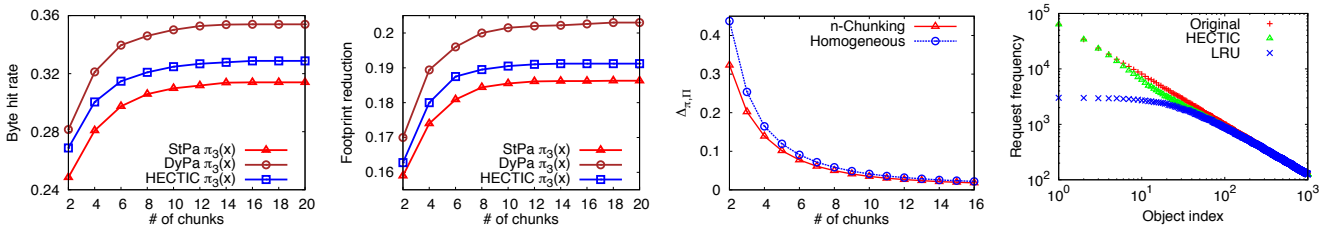
	level 0	level 1	level 2	level 3
HECTIC (1 GB)	0.3252	0.3207	0.3197	0.3249
HECTIC (2 GB)	0.4462	0.4427	0.4413	0.4489
LRU (1 GB)	0.3673	0.1784	0.1002	0.0647
LRU (2 GB)	0.4792	0.1936	0.0954	0.0512

TABLE II: Average BHR on each level of the distribution tree for both HECTIC and LRU with two cache sizes. Level-0 nodes refer to the leaf-nodes. The variances in all the cases are less than 10^{-4} therefore are not presented in the table.

B. HECTIC on a Regular Distribution Tree Topology

We first compared HECTIC against the dynamic and static optimal caching strategies from Section IV-A on a 4-level binary-tree topology with parameters as in Section IV-B. Because HECTIC is an en-route caching scheme, we restricted the optimal strategies in being able to retrieve data only from one hop away. Fig. 5a and 5b depict BHR and FPR respectively, and shows that HECTIC outperforms static StPa and reaches about 90% of the performance of the dynamic DyPa. Fig. 5c depicts the change in $\Delta_{\pi,\Pi}$ with increasing number of chunks. In line with the results of Figs. 5a and 5b, we observe that partial caching benefit becomes practically negligible beyond 8 chunks. One thing worth pointing out is that the number of chunks depends on the file size. Recall the conclusion 2 in Section III-C stating that the minimum number of chunks for a given performance target increases linearly as a function of file size. *From conclusion 2, we can further extrapolate that the percentage of a chunk constituting of the target file will decrease proportionally as file size increases.* For example, in our case, 8 chunks are enough for most Youtube videos which are practically around 10 MB, and each chunk constitutes about 10% of the file. Whereas for large files (e.g. high-definition videos), more chunks are certainly needed. Empirically, we notice in our evaluations that chunks smaller than 1 MB only bring marginal benefit on caching for most video files that are over 10 MB. Therefore, for a 1 GB video file, we need about one thousand 1-MB chunks, each of which constitutes only 0.1% of the file.

Fig. 5d plots the popularity distribution of the received requests on *level-1 nodes*, i.e., the next-hop nodes of edge-routers. As we can see, simple LRU renders the bending head in the popularity distribution, indicating severe filtering effect. In contrast, for HECTIC the observed popularity in upstream



(a) BHR as a function of # of chunks. (b) FPR as a function of # of chunks. (c) Δ as a function of # of chunks. (d) Filtering effect on level-1 nodes.

Fig. 5: (a,b) Performance comparison of HECTIC, StPa, and DyPa on the tree topology. (c) Change in Δ with increasing number of chunks. (d) Popularity distribution at level-1 nodes.

nodes is almost identical to the original distribution. The resistance to popularity deformation leads to high utilization of upstream caches, which further explains the reason why HECTIC outperforms other strategies.

Table II presents the average BHR of the nodes on the same level in a distribution tree where level-0 nodes refer to the leaf-nodes and level-3 node refers to the root. The results provide numerical explanation of Fig. 5d by showing how filtering effect impacts the actual caching performance. As moving farther from the edge to the root node in level 3, LRU's BHR quickly degrades and drops eventually to 5-6% due to the severe filtering effect. The same trend holds also for bigger cache size; first level serves a significant fraction of the requests from the cache and upper levels experience miss for most of the requests. On the other hand, HECTIC's BHR remains the same across all the levels regardless of the node position and cache configuration.

In other words, HECTIC spreads the most popular content along the path from the CP to the edges to ameliorate the filtering effects by increasing cache utilization in upstream nodes. The immediate intuition is that such operation may increase the average hops to deliver the content. However, considering most real-world networks have *small-world* property which indicates a short network diameter, the potential increase in fetching time is negligible in practice, and can be easily outrun by the gain from the reduction in intra- and inter-ISP traffic.

C. Evaluation with Realistic Settings

We verified HECTIC's performance in more realistic settings by comparing it on a testbed against several other caching strategies from literature. We used four ISP router-level topologies from Rocketfuel [23] project: Exodus, Sprint, AT&T and NTT. We only present results on Sprint network due to space limitations; other networks yielded similar results. All the experiments are performed on our department cluster consisting of 240 Dell PowerEdge M610 nodes equipped with 2 quad-core CPUs, 32GB memory, and connected to a 10-Gbit network. All nodes run Ubuntu SMP with 2.6.32 kernel.

We connected the server to the router with the highest degree and randomly connected clients at 30% of the routers. Each router is equipped with cache size of 3GB. Link cost and latency between two routers were set according to the topology traces, which reflects the realistic values.

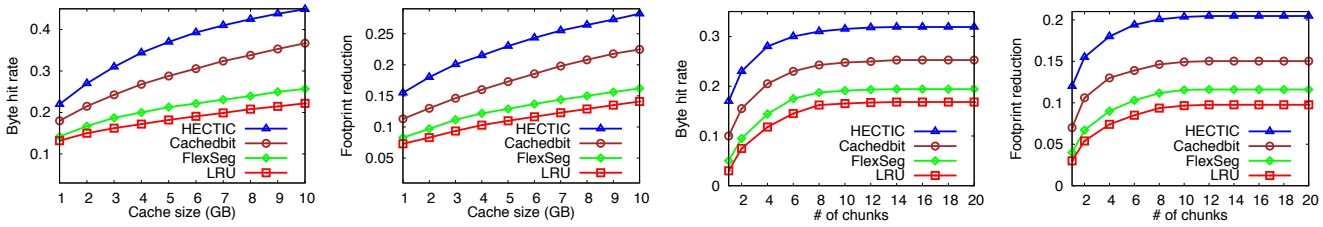
We used the gravity model for generating the traffic patterns. In the gravity model, we first map a client to the city according to its access router and traffic from the client is proportional to the city's population. The client in the city with the smallest population sends out 1 million requests and traffic from other clients is scaled up proportionally based on the city population.

For content, we used the real trace from Cha et al. [24]. We selected Youtube Entertainment Category trace which contains 1,687,506 objects. The trace contains video id, length, views, rating and etc. For other content parameters, we set based on investigations on Youtube by [14], [25]. The aggregated video size is 12.87 TB. Work in [24] showed that videos' encoding rates are similar, thereby it is reasonable to assume the file size is proportional to the video length. In [14], Cheng et al. showed that the average file size is 8.4 MB. Based on these results, we set the video size proportional to their length.

To the best of our knowledge, there is no publicly available data about file internal popularity. Therefore, we use a Weibull distribution to model file internal popularity. We assume user behavior on different videos differs. To model a heterogeneous access pattern, λ and k are drawn uniformly from $[1, 5]$ and $[0.5, 5]$ respectively, which models the various situations of how popularity varies within a file. All the results are calculated as the arithmetic average of 50 experiments. In each experiment, the client access routers are randomly re-selected to guarantee the results are robust and representative.

We chose another three caching algorithms to compare against HECTIC. LRU is a simple caching strategy with no special admission policy and LRU as replacement policy. Cachedbit is the algorithm presented in [18] and it uses the simple admission policy described in Section V-A and LRU as replacement. FlexSeg is a partial caching algorithm presented in [26] where it was also shown to have better performance than other partial caching algorithms.

Fig. 6a and 6b show how the performance of three caching strategies change as the cache size increases from 1 GB to 10 GB. Fig. 6c and 6d show performance change with the increasing number of chunks and keeping the per-cache storage at 3 GB. FlexSeg performs rather poorly, barely beating the simple LRU-based solution, whereas HECTIC shows very good performance. Compared with LRU, FlexSeg has a fine-grained policy to capture the user request pattern. Hence, it performs slightly better, especially in the case of a



(a) BHR as a function of cache size. 8 chunks per file.

(b) FPR as a function of cache size. 8 chunks per file.

(c) BHR as a function of # of chunks. 3 GB storage per node.

(d) FPR as a function of # of chunks. 3 GB storage per node.

Fig. 6: Performance evaluation of different in-network and partial caching strategies on Sprint network.

single cache. However, FlexSeg is designed for edge caches instead of a cache network. Like LRU, it suffers from very low utilization of the aggregate cache storage along the path, which accounts for its poor performance.

Conclusion: Naive chunking at the server and simple caching in the network outperform complex partial caching algorithms running in the network.

VI. Related Work

Caching per se has been extensively studied in mainly two different contexts, namely conventional IP network and information-centric network.

In conventional IP network, most prior work on partial caching concern video streaming services [26]–[34]. However, a distinction can be drawn between the work which assume a priori knowledge on content internal popularity, and which obtain such information during the algorithm runtime. For the first category with a prior knowledge, [27] proposes prefix caching which caches the initial frames to reduce the start-up latency at client side. Some others [28] are also developed on the similar idea that the earlier segments are more popular than the later ones. [29] further empirically studied the internal popularity of videos on a streaming website to verify aforementioned assumption. On the other hand, recent evidence [7] clearly shows that internal popularity can be arbitrary mix of non-continuous portions.

For the second category without such a priori assumption, [26], [30], [32]–[34] try to infer the content popularity from the request sequence. [30] proposes using two independent caches with different caching policies to improve hit rate. [33] utilizes lazy segmentation to determine the segment size on the fly. Similarly, flexible segmentation policy (FlexSeg) in [26] also allows runtime segment size, but the segment size is a function of both frequency and recency of accesses. [31] adjusts the coding rate dynamically to improve user experience.

In information-centric network, content is preprocessed before distribution and the system usually works at chunk-level. According to our knowledge, no research work has ever been done on partial caching and chunking analysis. However, the extensive work [9], [10], [15], [18], [19], [21], [35]–[39] on evaluating and modeling in-network caching deepened the understanding on cache networks and paved the way for our work. Close to our analysis on caching system, [15], [19], [35]–[37] also focus on the performance modeling and discuss the optimal content placement in cache networks. [18], [38], [39] further propose low-complexity heuristic similar to our HECTIC design, but chunking and filtering effects are not well studied. From practical perspective, [9], [10], [21] carried out thorough reality check of ICN design, showing content router is able to manage large amount of small chunks at line-speed. A recent work [40] shows that optimal on-path caching only slightly outperforms a simple edge-caching policy as contents are pushed closer towards the content consumers with increasing cache capacity. Regarding the performance metrics for ICN caching, [41] discusses that optimal caching schemes accounting for the *network-centric metrics* – cache hit rate and hop count – may not be the optimal solution if *user-centric metric*, i.e., content download delay, is the primary performance metric. As mentioned, though the caching mechanism has been intensively studied in various context, the content itself and its proper chunking scheme was largely overlooked by the community.

VII. Conclusion

In this paper, we study the partial caching by first identifying the origin of the partial caching benefit. Next, we propose a way to quantify the benefit of chunking and illustrate its relation to the actual performance changes. Based on this analysis, we present the optimal n-Chunking algorithm and compare it with homogeneous chunking. To evaluate the partial caching, we also develop an optimization model which can be used to calculate the upper bound of in-network caching performance. We devise a low-complexity heuristic, HECTIC, which performs close to the optimal solutions on simple topologies and outperforms existing caching solutions on realistic topologies. To the best of our knowledge, our paper is the first work in the ICN literature to provide a thorough theoretical analysis and empirical evaluation of chunking effects on caching performance. Our analytical and experimental results show that complex partial caching algorithms do not perform as well as a naive homogeneous chunking at the server combined with a simple utility-based caching strategy.

References

[1] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York, NY, USA: ACM, 2008, pp. 219–230.

[2] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, 1999, pp. 126–134.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th ACM Conext*. New York, NY, USA: ACM, 2009, pp. 1–12.

[4] Publish/Subscribe Internet Routing Paradigm, "Conceptual architecture of psirp including subcomponent descriptions. Deliverable d2.2, PSIRP project," , August 2008.

[5] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, 2007.

[6] C. Dannewitz, "Netinf: An information-centric design for the future internet," in *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, 2009.

[7] K.-W. Hwang, D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, V. Misra, K. K. Ramakrishnan, and D. F. Swayne, "Leveraging video viewing patterns for optimal content placement," in *NETWORKING 2012*. Springer, 2012, pp. 44–58.

[8] S. Vanichpun and A. M. Makowski, "The output of a cache under the independent reference model: where did the locality of reference go?" *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 295–306, Jun. 2004.

[9] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *Proceedings of the Re-Architecting the Internet Workshop*, ser. ReARCH '10, 2010, pp. 5:1–5:6.

[10] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*, ser. ICN '11, 2011, pp. 44–49.

[11] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.

[12] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. McGraw Hill, 2006.

[13] S. Choi, S. Cha, and C. Tappert, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, vol. 8, no. 1, pp. 43–48, 2010.

[14] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, june 2008, pp. 229 –238.

[15] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable icn," in *Proc. of the ACM SIGCOMM 2013 Conference on SIGCOMM*, New York, NY, USA, 2013, pp. 147–158.

[16] C. Williamson, "On filter effects in web caching hierarchies," *ACM Trans. Internet Technol.*, vol. 2, no. 1, pp. 47–77, Feb. 2002.

[17] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao, "On the intrinsic locality properties of web reference streams," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, 2003, pp. 448–458 vol.1.

[18] W. Wong, L. Wang, and J. Kangasharju, "Neighborhood search and admission control in cooperative caching networks," *IEEE GLOBECOM 2012, Anaheim, California, USA*, Dec. 2012.

[19] L. Wang, S. Bayhan, and J. Kangasharju, "Effects of cooperation policy and network topology on performance of in-network caching," *IEEE Communication Letters*, vol. 18, no. 4, April 2014.

[20] V. Sourlas, L. Gkatzikis, P. Flegkas, and L. Tassiulas, "Distributed cache management in information-centric networks," *Network and Service Management, IEEE Transactions on*, vol. 10, no. 3, pp. 286–299, September 2013.

[21] G. Rossini, D. Rossi, M. Garetto, and E. Leonardi, "Multi-terabyte and multi-gbps information centric routers," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 181–189.

[22] PARC, "Ccnx 1.0 wire format," 2014.

[23] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proceedings of ACM SIGCOMM*, 2002.

[24] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 1–14.

[25] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 15–28.

[26] U. Devi, R. Polavarapu, M. Chetlur, and S. Kalyanaraman, "On the partial caching of streaming video," in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, ser. IWQoS '12, 2012, pp. 4:1–4:9.

[27] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *INFOCOM'99*, vol. 3, 1999, pp. 1310–1319 vol.3.

[28] S. Jin, A. Bestavros, and A. Iyengar, "Accelerating internet streaming media delivery using network-aware partial caching," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 153–160.

[29] J. Yu, C. T. Chou, X. Du, and T. Wang, "Internal popularity of streaming video and its implication on caching," in *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, vol. 1, 2006, pp. 6 pp.–.

[30] K.-L. Wu, P. Yu, and J. Wolf, "Segmentation of multimedia streams for proxy caching," *Multimedia, IEEE Transactions on*, vol. 6, no. 5, pp. 770–780, 2004.

[31] R. Rejaie and J. Kangasharju, "Mocha: a quality adaptive multimedia proxy cache for internet streaming," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '01. New York, NY, USA: ACM, 2001, pp. 3–10.

[32] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 6, pp. 1447–1460, 2008.

[33] K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of the 10th international conference on World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, pp. 36–44.

[34] S.-H. Park, E.-J. Lim, and K.-D. Chung, "Popularity-based partial caching for vod systems using a proxy server," in *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, ser. IPDPS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 115–.

[35] J. Ardelius, B. Grönvall, L. Westberg, and A. Arvidsson, "On the effects of caching in access aggregation networks," in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ser. ICN '12. New York, NY, USA: ACM, 2012, pp. 67–72.

[36] I. Psaras, R. Clegg, R. Landa, W. Chai, and G. Pavlou, "Modelling and evaluation of ccn-caching trees," in *NETWORKING 2011*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6640, pp. 78–91.

[37] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie, "Optimal cache allocation for content-centric networking," in *IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–10.

[38] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ser. ICN '12, 2012, pp. 55–60.

[39] J. M. Wang, J. Zhang, and B. Bensaou, "Intra-as cooperative caching for content-centric networks," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ser. ICN '13, New York, NY, USA, 2013, pp. 61–66.

[40] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at "the edge" of information-centric networks," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. INC'14. ACM, 2014, pp. 47–56.

[41] M. Badov, A. Seetharam, J. Kurose, V. Firoiu, and S. Nanda, "Congestion-aware caching and search in information-centric networks," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. INC '14, New York, NY, USA, 2014, pp. 37–46.