# A Neighborhood Aware Caching and Interest Dissemination Scheme for Content Centric Networks

Amitangshu Pal and Krishna Kant
Computer and Information Sciences, Temple University, Philadelphia, PA 19122
E-mail:{amitangshu.pal,kkant}@temple.edu

***Abstract*—Content-Centric Networking (CCN) is a promising framework for the next generation Internet architecture that exploits ubiquitous in-network caching to minimize content delivery latency and reduce network traffic. In this paper, we introduce a neighborhood aware mechanism for content caching, named *Neighborhood Aware Caching and Interest Dissemination (NACID)* that accounts for the popularity of contents and how close the content copies are in the neighborhood. We have adopted a Bloom Filter based dissemination of caching information in the neighborhood so that its overhead remains small. Given the neighborhood cached contents, the proposed scheme decides when and how to handle the additional caching of content and its eviction. Simulation results show that NACID performs substantially better than existing CCN caching policies; it increases the cache hits by up to $\sim$5 times and decreases the number of hops required to access the content by up to $\sim$57%. We also study different heterogeneous cache memory allocation strategies and show that much simpler homogeneous allocation strategies work almost as well.**

***Index Terms*—Content centric networks, Caching, Interest dissemination, Content popularity, Zipf distribution, Bloom filter.**

## I. Introduction

The tremendous growth of Internet traffic in the recent past has led to intensive research into alternate, more scalable architectures. Based on Cisco's VNI report [1], the Internet traffic volume has increased eight times in the last five years. The annual traffic volume is anticipated to increase by 29%. Among the Internet traffic, the video traffic itself accounts for 86% of all the IP traffic in 2016 [1], which will continue to grow due to the growing demands for bandwidth-intensive services such as high definition Video on Demand (VoD) or time-shift TV services [2].

Most of this traffic is generated by content retrieval applications. This compels the Internet designers to shift from the sender-driven end-to-end communication paradigm to receiver-driven content retrieval paradigm [3]. The emerging information-centric networking (ICN) [4] architectures are based on the observation that unlike the classical Internet architecture that is based on the addresses of nodes and routing between these addresses, the new Internet architecture should instead focus on information availability and demand. That is, a piece of information should be identified by its own characteristics rather than where it resides, and its spread in the network should be controlled by its demand and QoS requirements (e.g., hard real time, transactional, etc.). These ideas have been investigated generally under the name content-centric networking (CCN) [5], [6], [7] and more specifically under the NSF FIA project called Named data networking (NDN) [8]. Thus a key concern in ICN/CCN/NDN is where to host the content most efficiently based on the demands that may be changing dynamically. This is done by using a publish-subscribe model to match the demand with availability and a dynamic *caching* mechanism to move the hosting of the content closer to the demand points.

In-network content caching has been studied extensively in the current IP-based networks also, such as Web service, P2P, CDN [9], [10], [11]. However, such mechanisms do not apply directly to CCN caching due to the lack of unique and universal content name. For example, in Web caching if two copies of the same content are placed in different servers of different content providers, different URLs are used to identify and access the content [3]. This makes the existing Web caching or CDN caching unsuitable for CCN caching.

Caching of contents in CCN is also well studied [12], [13], [14], [15]; however, most of these schemes use the notion of *path caching*. That is, if the content is located at an origin node $x$, and node $y$ requests it, most schemes cache it along the path, although the decisions about which nodes cache it varies. For example, the content may be cached at every node in the path, at the next node down from the last caching place, etc. In contrast, we propose here a *Neighborhood Aware Caching and Interest Dissemination (NACID)* scheme where the caching decision is made based on whether any copy of the content exists in the neighborhood of the requesting node, and how far the requester needs to go to fetch the content. We link this cost to the predicted demand for the content and its obsolescence rate. The simplest characterization of the neighborhood size can be in terms of the number of hops from the requesting node; however, more sophisticated metrics such as a given delay limit can also be considered. Different CCN links may have different costs (capacities, traffic volumes, delay etc.), thus in NACID the nodes need to consider the routes to their neighborhood content stores along with their route cost, before evicting the content.

The main contributions of this paper in enabling neighborhood aware caching are as follows. First, we present an efficient Bloom Filter based dissemination mechanism in the neighborhood so that its overhead remains small. We also

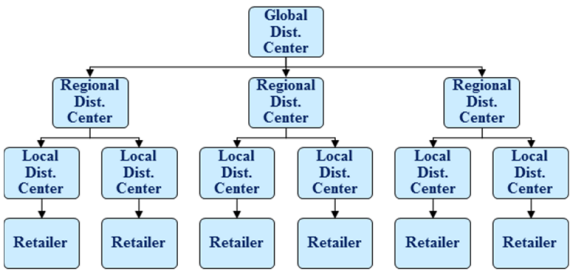



Fig. 1. An abstract model of PCDN.

propose a *two-level caching architecture* in NACID where the first level makes caching decisions *synchronously* (i.e., driven by the arrivals of the newer contents), whereas the second level does so *asynchronously* (i.e., done periodically as a housekeeping activity). These two operations need to be done carefully else they could result in thrashing, bandwidth waste, and additional delays. We develop a two-level (short-term and long-term) caching scheme to address these issues. The paper quantifies the advantages of the proposed approach via extensive simulation studies that show that the NACID increases the cache-hit ratio up to ∼5 times, whereas the hop-reduction percentage goes down up to ∼57%. NACID can be used for both inside an ISP as well as across the ISPs with some peering arrangements.

The outline of this paper is as follows. We first discuss the key motivation behind developing the NACID architecture in section II. Section III-IV propose the system model, content popularity distribution and network architecture assumed in our scheme. Section V introduces the proposed neighborhood aware caching scheme and describes the operation and interaction of the two-level caching mechanism. Section VI shows the simulation comparison of NACID against other well-known existing schemes. Section VII studies the performance of NACID from some real Youtube traces from a campus network. Relevant literature and discussions are summarized in section VIII. Finally, we conclude the paper in section IX.

## II. Motivation behind NACID

Interestingly the key motivation behind this work stems from our recent efforts for building an efficient Perishable Commodity Distribution Networks (PCDN) [16], [17]. We observe that a significant amount of synergies exist between the PCDN logistics and the CCN architecture. Fig. 1 shows a typical PCDN logistics architecture, which also works as a *producer-consumer* model similar to CCN. In PCDN the commodities move from "source" to "destination" end-points, the former being farms and manufacturing/assembly plants, and the latter retailers and other large customers (e.g., restaurants, hospitals), though there is generally no transportation in the other direction. Commodities flow from source to destination via a number of intermediate points which include local, regional, and global distribution centers as shown in Fig. 1. These nodes can store full or empty containers, change container contents (by removing, adding, or exchanging packages), load/unload containers on carriers, handle damage/misdelivery, etc.

Perishability is a key QoS driver in PCDN. Products often deteriorate in quality or in value/usefulness as a function of flow time through the logistics system. The deterioration as a function of time $t$ can be described by a non-decreasing function that we henceforth denote as $\zeta(t)$. In general, $\zeta(t)$ is linear for fruits or vegetables and exponential for fish/meat. In CCN too the value of information declines steadily with the delay incurred. One significant example of perishable content is the breaking news stories that are typically updated periodically based on the new developments. The older versions get progressively less useful, and at some point worthless.

In PCDN the popular commodities are stored and ordered in large quantity compared to the others, which again is identical to the caching of more popular contents against the rare ones. Thus proactively storing a popular commodity in logistics is often beneficial compared to the unpopular ones. In PCDN a sudden demand at a retailer can be satisfied from some nearby distribution points or retailers (instead of bringing all the way from the "source"). This is technically known as *lateral distribution* in logistics. CCN has similar characteristics in that the content can be fetched from some neighboring cache, rather than bringing from the actual source server.

The above producer-consumer based PCDN model has three key fundamentals concepts that we want to capture in NACID. **First** is the perishability characteristics of Internet contents which need to be stitched into the network model so that the users get up-to-date contents upon request. **Second** is the notion of dynamic popularity of the contents, and we use it to model a benefit function of caching (or not) the contents in CCN routers. **Third** is to model the lateral transfer from neighborhood caches, which results in neighborhood aware caching in CCN context. We next discuss these points in section III–V.

## III. The System Model

### A. Content Popularity Distribution

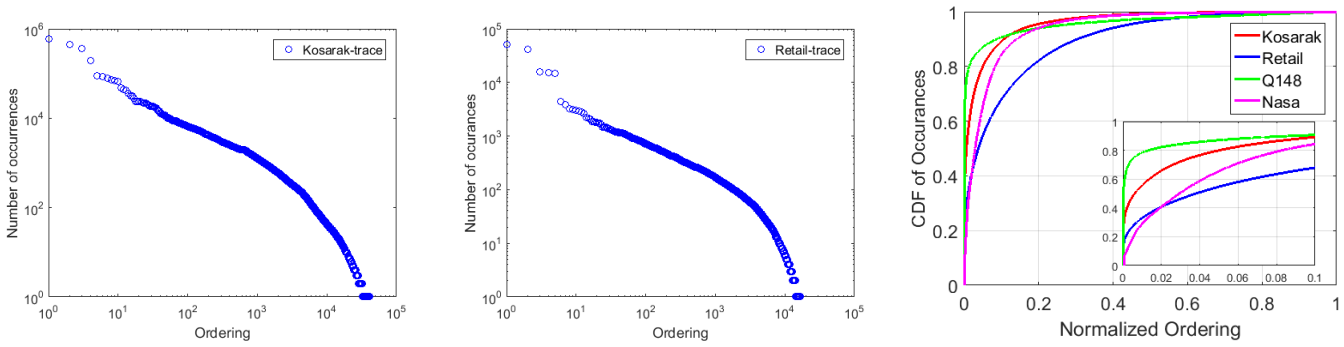
In CCN the content popularity is determined by how often a piece of content is requested. Recent studies [18], [19] show that the users are attracted by only few contents, while others are accessed rarely. In fact a significant portion of the contents are one-timers. Therefore, the content popularity is commonly modeled with the Zipf distribution function, which states that the size of the $i$-th largest occurrence of an event is inversely proportional to some power of its rank. In a Zipf distribution, out of the population of $\mathcal{M}$ contents, the frequency of the $i$-th content is given by

$$f(i,\alpha,\mathcal{M}) = \frac{\frac{1}{i^\alpha}}{\sum_{j=1}^{\mathcal{M}} \frac{1}{j^\alpha}} = \frac{\frac{1}{i^\alpha}}{\mathbf{H}_{\mathcal{M},\alpha}} \tag{1}$$

where $\alpha$ is the Zipf exponent and $\mathbf{H}_{\mathcal{M},\alpha} = \sum_{j=1}^{\mathcal{M}} \frac{1}{j^\alpha}$ the generalized harmonic number of order $\alpha$. In literature the range of $\alpha$ is varied from 0.6 [20] to 2.5 [21].

By taking logarithmic values on both side in equation(1), we obtain

$$\log f(i,\alpha,\mathcal{M}) = \log\left(\frac{1}{\mathbf{H}_{\mathcal{M},\alpha}}\right) - \alpha \log i \tag{2}$$




(a) (b) (c)

Fig. 2. Frequency of content accesses versus content ranking for (a) `Kosarak` ($\alpha$ = 1.99) and (b) `Retail` ($\alpha$ = 1.55) traces [22]. (c) Cumulative distribution of content demands vs content ranking for different traces.

which means the distribution function is a linear in a logarithmic scale. When $\alpha = 0$, it corresponds to a uniform distribution. When $\alpha > 1$, the frequency of the less popular contents tend to drop quickly.

To illustrate the effects of Zipf based popularity distribution, we use two real datasets, named `Kosarak` and `Retail`, that have been widely used in the data-mining literature and follow power law distribution. `Kosarak` is a click-stream dataset of a Hungarian online news portal that has been anonymized, and consists of transactions, each of which is comprised of several integer items. `Retail` is a retail market based data obtained from a Belgium store. Figs. 2(a)-(b) show the number of times a content has been accessed versus the ordering of the content in the trace. Both figures show a roughly linear plot (in the log-log scale) for frequently accessed items. The tail behavior is usually different and can be captured more accurately with more complex distribution functions, but this may not be necessary.

Fig. 2(c) shows the CDF of occurrences of the top $r\%$ contents in four traces (`Kosarak`, `Retail`, `Q148`, `Nasa` obtained from [22]). From this figure we can observe that top 1% of the contents are accessed for about 80% of the time in case of `Q148` traces, whereas varies in between 30-60% for others. Whereas the top 10% of the contents are accessed more than 60% of the time in all traces. Thus correctly identifying the hot contents in CCN is crucial for caching decisions. We thus model some content popularity prediction schemes in section IV, which are used in content caching in section V.

### B. Content Popularity vs Freshness

The contents served by the Internet are increasingly dynamic in that they are frequently updated. For example, news stories become stale sooner compared to reality shows or movies since they are constantly updated. Also, different types of news have different update rates and useful life, e.g., those concerning a fast moving disaster vs. normal events. The notion of *Content Freshness* can be used to capture this aspect and is crucial in CCN to serve the clients with up to date information [23]. To incorporate content freshness in NACID, we consider a few CCN nodes, defined as Repositories (*Repo* in short) that are deployed in different regions, with larger storage compared to the routers. Such nodes act similar

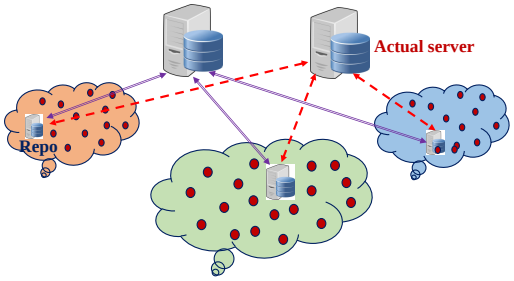



Fig. 3. The proposed CCN architecture.

to the content delivery routers, and are supported in NDN architecture [8]. These nodes work as content servers in their neighborhood regions, as shown in Fig. 3.

Such an architecture is again very similar to the PCDN architecture shown in Fig. 1, where the local, regional and global distribution centers correspond to the content routers, Repos and actual server respectively. In such an architecture Repo periodically/occasionally consults with the original servers to check whether certain contents are stale and/or expired. Whenever it finds a change in some contents, it informs its neighboring routers using a Bloom Filter to purge those contents. Further requests for those contents would then be directed towards the Repo, which would send fresh and consistent contents. In this paper, we only consider exploring caching and content interest dissemination mechanism for fetching the contents from a Repo to a number of neighboring content routers, whereas the details of the message passing in between the Repos and the actual servers for maintaining fresh contents is beyond the scope of this paper.

## IV. Content Popularity Prediction Model

The popularity of a content varies from region to region. For example, a regional news or sport may be popular within a region but will be rarely accessed by the users in other regions. Thus the popularities of programs with regional dialects or importance greatly vary spatially and temporally. To predict this dynamic and regional popularity, we first consider a few well-known time-series prediction schemes as mentioned below. These prediction models will run at each CCN router independently to capture the *regional* variation of content popularities.

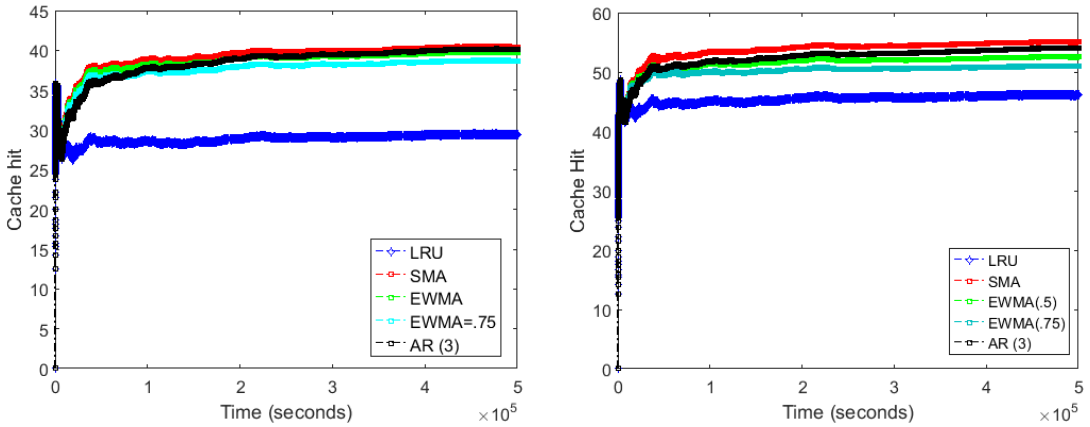



Fig. 4. Comparison of LRU and other popularity prediction based content caching schemes with (a) cache size = 100, and (b) cache size = 500 for `Kosarak` trace.

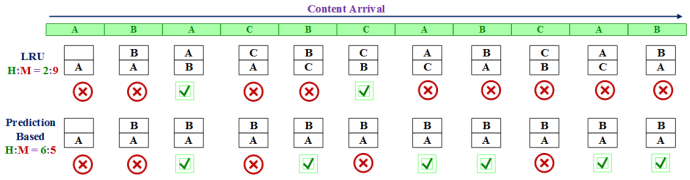



Fig. 5. An illustrative example for comparing LRU and popularity prediction based caching.

**Simple moving average model (SMA):** Let $Y_t^c$ denotes the local demand of content-$c$ (i.e. the number of accesses) at a router at time $t$. Then the SMA model predicts the demand for the next time slot as simply the average from the actual demands from the last $m$ slots [24]. That is, $\hat{Y}_{t+1}^c = \left(Y_t^c + Y_{t-1}^c + \ldots + Y_{t-m+1}^c\right)/m$.

**Exponentially weighted moving average model (EWMA):** EWMA uses a simple exponential smoothing for prediction, i.e. $\hat{Y}_{t+1}^c = \alpha Y_t^c + (1-\alpha)\hat{Y}_t^c$, where $\alpha$ is the smoothing constant in between 0 and 1. EWMA is the most widely used time series prediction model.

**Autoregressive (AR) model:** An Autoregressive (AR) model is one of the most popular methods for modeling and predicting future values of a time series [25]. Given the past demands of $c$, an AR model of order $p$ is defined as:

$$Y_t^c = \sum_{i=1}^{p} \beta_i Y_{t-i}^c + \varepsilon_t \tag{3}$$

where $\beta_1, \ldots, \beta_p$ are the parameters of the model and $\varepsilon$ is a white noise error term. The error terms, $\varepsilon_t$, are generally assumed to be Gaussian i.i.d. random variables with zero mean and constant variance.

We evaluate the effectiveness of prediction based caching against the *least recently used (LRU)* based caching using `Kosarak` [22]. Since LRU replaces the least recently used content, it can be thought of as a popularity prediction mechanism based on the stack distance estimate. Fig. 4(b)-(c) show the comparison of LRU, SMA, EWMA, and AR (ties due to identical $\hat{Y}$ are broken based on the content recency) with cache size 100 and 500 respectively. The contents are assumed to arrive at one per second, whereas, each slot spans 1000 seconds. For SMA we assume $m = 5$ for Fig. 4. For AR model we assume $p$ and $q$ to be 3. From these figures we can observe that the above content access prediction based schemes improve the cache hit by ∼10-12% in comparison to LRU.

Similar improvements are also observed with other well-known trace files [22], [26]. Such improvement is explained in Fig. 5, where A and B are two popular contents and C is relatively less popular. Also assume that a cache can store two contents at any time. In such situation we can observe that the LRU strategy performs poorly as compared to a prediction based scheme that can predict the popular contents (i.e. A and B) and store them irrespective of their recency. Thus the popularity prediction based caching scheme experience 6 hits as opposed to just 2 hits in case of LRU. The success of the scheme strictly depends on how accurately and quickly it can distinguish the popular contents A and B as opposed to C.

We can also observe from Fig. 4 that all the prediction schemes perform almost similar. The reason is that all these schemes may vary in terms of their prediction accuracy, but can distinguish the popular contents as opposed to the less popular contents almost identically. Because of this reason, the hit ratio is similar for all these schemes. We thus use the SMA based popularity prediction model for the rest of the paper, for simplicity. Other complicated prediction models (like EWMA or AR) can also be used, however, we consider SMA mainly because it is simple, lightweight and can be easily implemented in CCN routers. Such popularity predictions are useful for taking effective caching decisions as discussed in section V.

Now let us analyze the memory overhead for tracking the demands corresponding to every content in SMA. The number of total content name prefixes in today's Internet is around 100 million [27]; however, there is a quite a bit of locality in Internet traffic, and every tier1 router carries a very tiny fraction of the possible 100M contents. This cross section is further narrowed in case of contents: most items (i.e., videos, movies, news stories, etc.) are only of local, regional, and country/language specific interest. Furthermore,

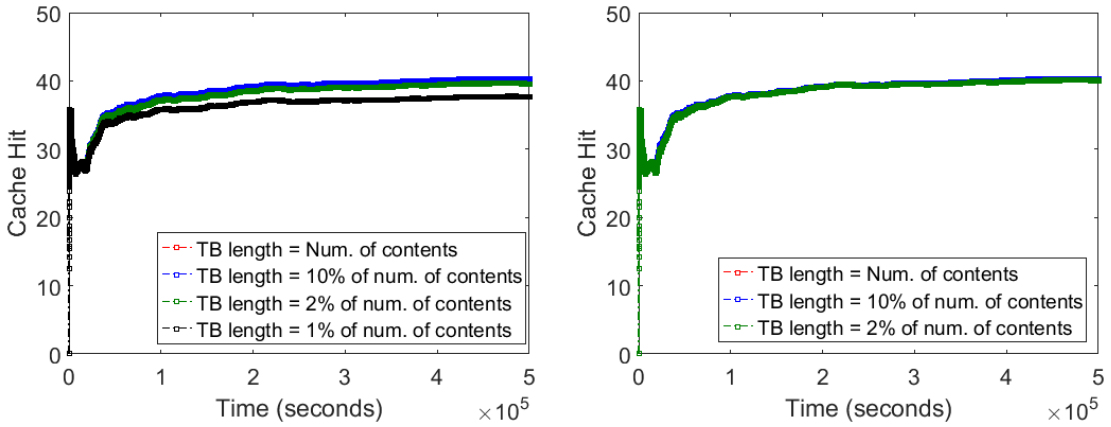



Fig. 6. Effect of limited TB length on SMA performance with (a) TB-LRU and (b) TB-LFU.

the time zone differences across the prefixes provide further temporal locality of accesses. However, the most important characteristic to limit the memory consumption is the highly skewed (e.g., Zipf like) access pattern. That is, by keeping only a few percent of the most popular objects, we should be able to capture most of the repeat accesses. This is easily accomplished by treating the *tracking buffer (TB)* as a cache using any replacement policies.[1]

Let us assume that TB can store the demands of $L$ contents for $m$ slots, where $L$ is less than the number of contents, and $m$ is the length of SMA. For experimentation we discuss the performance of two replacement policies in TB. First, we consider a simple LRU policy for replacing the entries, when the TB is full. Notice that the number of entries in TB will be much more than that of the cache, and so using LRU for maintaining TB is not identical to that of caching. We call this scheme TB-LRU. The second policy is to replace the contents that has been accessed the least in the last $m$ slots, which we call "TB-Least Frequently Used" or TB-LFU policy. To ensure that the newly coming contents are not replaced immediately after been inserted in TB, we ensure that the contents that are accessed in the current slot are not replaced before the end of that slot, i.e. if a content is requested in $(t-1, t)$, then it will not be replaced before the end of $t$. Notice that in TB-LFU policy, the TB length should be atleast as large as the number of content accessed during a slot time. Such a requirement is not needed in the TB-LRU policy.

Fig. 6 shows the performance of these two policies with different TB lengths on `Kosarak` dataset. With both policies the hit ratio is hardly affected till the TB length is 2% of the total number of contents, i.e. nearly all of the repeated accesses are captured by using a TB size that is only 2% of the total number of unique items. In case of TB-LRU policy the cache hit deteriorates slightly when the TB length becomes 1% of the number of contents. Notice that we did not show this case for the TB-LFU policy, because of its requirement of having TB length more than the number of contents accessed in a slot time. To ensure this scheme to be beneficial that the TB length should be much larger than the number of contents a cache can store, so that the TB list can have a better view of the content access demands. Note that even when the TB length is only a small fraction of the number of contents, the scheme is still better than simple LRU based caching.

[1]This TB cache is different from and much smaller than the data cache; it only tracks the content ID's.

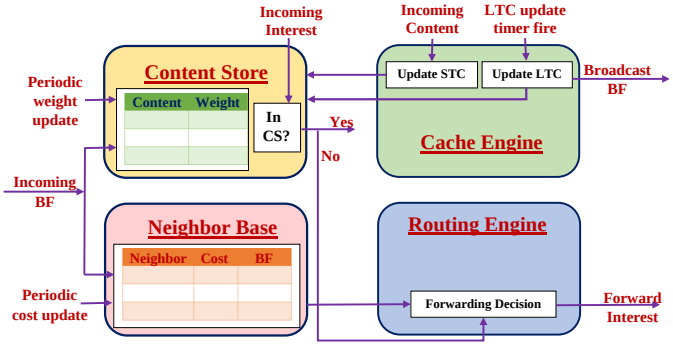



Fig. 7. The overall NACID architecture.

It is also worth commenting that the resources available at any Internet tier must be commensurate with the number of routing paths or contents handled by it. For example, suppose that we use 8B for content ID/item, 4B for access frequency/item, and 8B for pointers or other data structure elements. Also suppose that we keep history length of at most 10. This results in 16+40=56B per item. Thus, a tier1 router handling simultaneous 100M items would only need at most 112MB of memory (assuming 2% most popular item IDs are cached). This is insignificant for a tier1 router that is likely to have 64GB or more memory. Similarly, a much smaller regional router handling 1M items and using 5% caching to capture even less frequent items will still need only $\tilde{2}$.8 MB memory. As an example, the YouTube video traces collected from a gateway router at University of Massachusetts, Amherst [19] shows only 300K unique items over a period of two weeks. The authors in [28] have also mentioned around 500K unique videos over a two-week period from a leading online video content provider in China. Similar comments apply to the computing power required.

## V. Neighborhood Aware Caching and Interest Dissemination in NACID

We next introduce a *neighborhood aware* mechanism for content caching and information dissemination scheme for CCN. We assume that each CCN router is assigned a unique ID with a flat or hierarchical structure [29]. We also assume that the contents are divided into smaller *chunks* which are identified by their unique names or IDs. Compared to the previously studied schemes [12], [13], [14], [15] on path caching, in NACID the caching decision is made based on (a) where the content exists in the *neighborhood* of the requesting node, and (b) its predicted content demand and its obsolescence rate. The overall NACID architecture is shown in Fig. 7. The entire scheme is summarized below, by describing the two key modules, named *Cache Engine* and *Routing Engine* that run at each router.

### A. Cache Engine

The main challenge in enabling the neighborhood aware caching is the advertisement of the cached content-chunks, while keeping the overhead small. To address this issue, we propose a two-level caching scheme, as shown in Fig. 8. We assume that the entire cache/Content store (CS) of a node is divided into two levels, the upper level is the long-term cache (LTC) where the most useful content-chunks are cached. The rest is used to reserve the less useful chunks, and is known as short-term cache (STC). The STC cache is updated at each arrival of a chunk, to check whether the chunk is going to be cached or not. Occasionally the existing cache is *reshuffled*, where more useful chunks are transferred to the LTC and others are placed in the STC. This reshuffling can be done either periodically or when the the STC is changed significantly. After such an update, the information regarding the LTC chunks is broadcast up to a certain number of hops, which is defined as *broadcast range* $\mathcal{B}$. As the CCN content names are much complex and longer than IP addresses, we use Bloom filter (BF) to encode the presence of a content in a router's LTC.

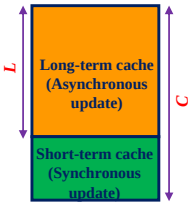



Fig. 8. Two level caching.

A Bloom filter is a hash-coding method used to represent a large set and at the same time supports membership queries on the set. The key difference between Bloom filters and traditional hash based representations of a set of elements is that the space required for Bloom filters is considerably reduced at the cost of permitting a small fraction of errors. Each content (key) is hashed using $k$ different hash functions and the resulting "hash positions" are updated to 1. When there is a membership query for a key (or content), if all $k$ hash positions of the key are set to 1, then a positive membership query is returned. While the false negative probability is zero, the false positive probability is a tunable parameter, which depends on the size of the filter. BF offers an efficient way to represent the set of cached chunks and takes $\mathcal{O}(1)$ time

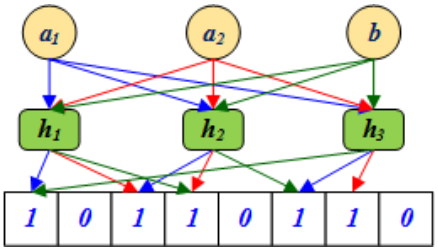



Fig. 9. A typical Bloom Filter.

to check whether a given chunk is within the set. A typical example of Bloom filter is shown in Fig. 9 where two contents $a_1$ and $a_2$ are inserted in a bloom filter by using three hash functions ($h_1$, $h_2$ and $h_3$) by setting the corresponding bit positions to 1. An illustration of a false positive scenario is also shown in this figure where the presence of content $b$ is wrongly inferred as the three hash functions map $b$ to the bit positions that are set to represent the presence of $a_1$ and $a_2$. In typical bloom filters elements can be added to the bloom filter, but cannot be removed.

The LTC cache remains unchanged throughout the *update interval* (i.e. the time in between two successive cache reshuffles). By keeping the LTC chunks unchanged within an interval, the BF broadcast is limited to one per interval. Note that the STC elements are not shared in the neighborhood. Given such a mechanism, it is easy to *reactively* cache the incoming chunk if it is not available in the close neighborhood.

When a new chunk arrives at a node, the STC decides which chunks (if any) should be replaced. The cache reshuffling, done periodically, chooses the most useful chunks to store in LTC and broadcast. For this, we define the benefit ($w_i$) of a chunk by including two factors: (a) cost-demand factor, which is the product of the predicted access demand $\hat{Y}_t$ and the cost $c_t$ to get it from the nearest neighbor, and (b) recency factor, which is inversely proportional to the time since last access. That is,

$$w_i = \zeta c_t \hat{Y}_t + \frac{\gamma}{\max(\Delta_i, \varepsilon)} \quad \left(\zeta \gg \frac{\gamma}{\varepsilon}\right) \tag{4}$$

where $\Delta_i$ is the difference between the current time and the time when a chunk was last encountered. The term $\varepsilon$ ensures that the second factor cannot be a dominating factor for very small $\Delta_i$. The simplest form of $c_t$ can be the number of hops to the nearest neighbor node; however, more sophisticated cost metrics such as link capacities, traffic volumes, delay etc. can also be considered. For our experiments, $c_t$ is calculated by the hop-counts. The intuition behind calculating $w_i$ is as follows: it is beneficial to cache a chunk that has (a) high demand $\hat{Y}_t$, (b) is cached in a router that is far away (i.e. high $c_t$), and (c) is recently encountered (i.e. low $\Delta_i$). In equation(4), $\zeta$, $\gamma$ and $\varepsilon$ are hyper-parameters with $\zeta \gg \frac{\gamma}{\varepsilon}$, which ensures that the cost-demand factor dominates while calculating the benefit of caching a chunk; when this factor is almost identical to some contents, then the ties are broken by using their recency.

With these, the general caching problem is described as follows. Assume that $y_i$ is the decision variable to check whether a chunk is going to be cached or not, and $s_i$ is the size of the $i$-th chunk. Then the problem is to choose certain

chunks from a set of $\mathcal{M}$, that can be accommodated in a cache size of $C$, which can be formulated as follows:

$$\text{Max} \sum_{i=1}^{\mathcal{M}} w_i.y_i \quad \text{subject to} \sum_{i=1}^{\mathcal{M}} y_i.s_i \leq C, \quad y_i \in \{0,1\} \quad (5)$$

The above problem is identical to the 0-1 Knapsack problem [30] in combinatorial optimization, which is proven to be NP-hard. We thus propose a greedy heuristic which is similar to the greedy knapsack solution, as described in Algorithm 1. The scheme first sorts the chunks in decreasing order of $\frac{w_i}{s_i}$ and then caches them sequentially until the cache space is filled up.

**Algorithm 1** Greedy caching

1: INPUT : Cache capacity $C$, benefits ($w_i$) and sizes ($s_i$) of chunks $i = \{1, 2, ..., \mathcal{M}\}$.
2: OUTPUT : Vector $y_i \in \{0, 1\}\ \forall i \in \{1, 2, ..., \mathcal{M}\}$.
3: Sort the chunks in decreasing order of $\frac{w_i}{s_i}$, i.e. $\frac{w_1}{s_1} \geq \frac{w_2}{s_2} \geq ... \geq \frac{w_{\mathcal{M}}}{s_{\mathcal{M}}}$;
4: Define $\ell = \min\{\xi \in \{1, ..., \mathcal{M}\} : \sum_{i=1}^{\xi} s_i > C\}$;
5: $y_i = 1$ corresponding to the chunks (1, 2, ..., $\ell - 1$) and 0 otherwise;

We note the following properties of our greedy algorithm:

*Observation 1:* If the cache size is much larger than the maximum chunk size, and $\max\{w_i\} << \sum_{i=1}^{\mathcal{M}} w_i$, then greedy algorithm approaches to the optimal solution.

**Proof:** The solution of Algorithm 1 and the continuous (or LP-relax) version of the knapsack problem differs by at most one element. The 0-1 knapsack problem is upper bounded by its LP-relaxation version, and Algorithm 1 differs from the LP-relaxation version by just one element. Thus in the limiting case of large cache, Algorithm 1 approaches to the optimal result, provided $\max\{w_i\} << \sum_{i=1}^{\mathcal{M}} w_i$.

*Observation 2:* When all chunks are of the same size, the greedy algorithm converges to the optimal solution.

In our simulations, we assume that all chunks are of equal sizes, which is generally assumed in literature [31]. The assumption can be justified as follows: for heterogeneous content sizes, the contents are split into chunks of identical sizes where each of them can be considered as individual contents. Such equal size chunks are used in Dynamic Adaptive Streaming over HTTP (DASH) protocol which usually splits each video content into several equal-sized chunks, as reported in [31].

Algorithm 1 is used asynchronously at the time of cache reshuffling, to keep the most useful chunks to LTC, whereas others go to STC. The same algorithm is used to reactively make the decision of caching (or not) the incoming chunks in STC depending on their benefits.

**Time complexity of maintaining the STC in case of identical content sizes:** The contents-chunks are placed in a MIN-HEAP data structure (depending on their benefits) for taking the caching decision efficiently. This ensures that in case of identical content-chunk sizes, this reactive mechanism just requires a benefit comparison between the newly arrived chunk and the chunk with least benefit in STC (i.e. at the root of the HEAP), and thus can be done at the line speed of the routers. If the newly arrived chunk is cached, the root of the HEAP is replaced by the new content and MIN-HEAPIFY is called to maintain the HEAP, which can be done in $\mathcal{O}(\log n)$ time.

### B. Routing Engine

Another component in Fig. 7 is the *Routing Engine* that forwards the Interest packets. The existing CCN mechanisms forward Interest packets towards the content server through the shortest path since they are unaware of the cached chunks in their neighborhood. Since our mechanism is aware of the caching (only LTC chunks) in the neighborhood via a Bloom Filter (BF) mechanism, the routing engine forwards the Interest packets towards the nearest (or least cost) cache instead. To calculate the cost among their neighbors, the nodes periodically exchange the updated link cost information (bandwidth, traffic volume, congestion, delay etc.) in their neighborhood. However, for simplicity we will only use hop-count as the cost-metric for our simulations. Each router maintains the cost information along with the broadcasted BF from its neighbors in its Neighbor table (or Neighbor base). Upon arrival of a new BF from any neighbor, this table is updated corresponding to that neighbor. This table is referred by the routers to forward the Interest messages towards the nearest cache. If no neighbor entry is available corresponding to a chunk, it is forwarded towards the repository.

The overhead of this scheme is that the routers need to store the bloom filters of all their neighbors in their Neighbor base. However in section VI-H we show that the additional memory overhead of this scheme is very limited for practical cache sizes and thus can be used in a realistic CCN environments. For further reducing the memory requirements other kinds of bloom filters (such as *compressed bloom filter* [32], *cuckoo filter* [33] etc. ) can also be used, details of which are beyond the scope of this paper.

One can also argue that the CCN routers can store the hash functions (like SHA-1, MD5 etc.) of their cached contents, and broadcast the hashes of *only* the entries that have changed. If the entries do not change too frequently this might be more efficient, whereas if they change frequently then the Bloom filter exchanges will be more efficient.

In both schemes, each node needs to store the compressed representation of the LTCs of all the nodes in its neighborhood. With hashing, the representation is in form of a table of hashes of all LTC entries. The size of the hash entry determines the collision probability. For example with a cache size of $10^6$ chunks and with a 32-bit hash function that spreads the values fairly evenly, the collision probability of the order of $10^{-3}$. This amounts to 4MB space per node. With Bloom filter, the size determines the false positive probability. As shown in Table I, with $10^6$ chunks and false positive probability of $10^{-3}$, the per node size is only 1.7MB. Thus, Bloom filter is more efficient from storage perspective; however, unlike the hashing solution, the entire filter must be transferred to convey the modifications in the LTC at a neighbor. In order to reduce the transfer overhead, it is possible to forego broadcast of a BF to the neighborhood if only a few entries have changed.

### C. Putting It Together

With these we next propose the overall procedure of NACID. If a CCN router is interested in a content-chunk that is not there in its cache, it first checks whether the chunk

is there in its neighborhood by consulting with the Neighbor Base. If it is not found in the Neighbor Base, the Interest is forwarded to the Repo. Otherwise the Interest is forwarded to the neighboring router with the least cost. The Interest packet carries the ID of the neighboring router that has the chunk. Along with the ID, the Interest packet also carries a setAggregate flag which is set to *true* by default (we describe the use of this flag shortly).

Each router receiving an interest should first check whether the requested chunk is present in its local cache by looking up the Content Store (CS) table. If there is a hit, the router forwards a copy of the chunk to the requester along the reverse path. Otherwise the router forwards the Interest towards the router/Repo whose ID is mentioned in the Interest packet.

The Pending Interest Table (PIT) is used to record the ongoing requests. When a router generates an Interest, each router in the path towards the destination adds an entry in its PIT. When the response comes back, this table is used for sending back the requested chunk through the reverse path towards the sources of the Interests. While forwarding the chunk back in the reverse path, the CacheEngine of the CCN routers determine whether to replicate the chunk in the STC based on the proposed caching strategy. Each Interest has an associated lifetime; its PIT entry is removed when the lifetime expires. When multiple Interest packets (with setAggregate = true) for the same chunk arrive at a CCN router, only the first Interest packet is forwarded whereas others are suppressed for reducing the network traffic.

Notice that the effectiveness of the forwarding mechanism depends on the BF size as well as its false positive probability. Due to the false positive probability, an Interest packet can be forwarded to a router $i$ that does not have the desired chunk. In that case router $i$ detects it and forwards the Interest packet to the Repo, with the setAggregate flag set to *false*. When a router receives an Interest with setAggregate = false, it forwards the packet towards the Repo instead of suppressing it. For example in Fig. 10 assume that $R_1$ sends an Interest packet with setAggregate = true to $R_3$ thinking that it stores a particular chunk. This Interest packet is forwarded by $R_2$, any other Interest packets with setAggregate = true that arrive at $R_2$ are suppressed. When $R_3$ receives the Interest packet, it checks its CS and realizes that the Interest is wrongly sent to it. It then forwards the Interest packet towards Repo with setAggregate = false. Whenever routers like $R_2$ receives such an Interest packet with setAggregate = false, it forwards it towards Repo instead of suppressing it.

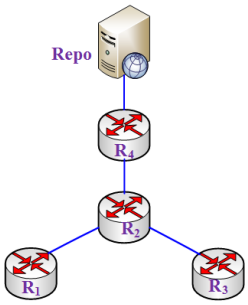


Fig. 10. An illustrative example.

Notice that NACID requires the information passing across the content routers, which can be addressed in two ways. First, NACID can be used freely within the scope of an ISP. This can be quite useful in itself because a physical region is often served by one (and at best a few) ISPs, and even when multiple ISPs are present, they often provide different services, each with their own unique content. Second, when two or more ISPs provide services that involve overlapping content in the same area, we can expect peering arrangements between them as such arrangements help both providers and customers. We also envision a finer grain resource sharing and access control in the future so that all ISPs can more effectively deal with congestion and slash-dot situations.

## VI. Simulation Results

We analyze our proposed CCN scheme using **CCNSIM** [34], which is an application-level simulator for content centric network based on OMNeT++. We assume a $10\times10$ grid topology consisting of 100 nodes. The content store (Repo), is at a corner of the grid. The content request of a requesting node is assumed to be Poisson with an arrival rate of one request/second. To keep the control overhead low, the update interval is assumed to be periodic with a period of 200 seconds. We compare our proposed scheme NACID with the following popular CCN schemes. The default replacement policy of the following schemes are assumed to be Least Recently Used (LRU).

**LCE:** Leave Copy Everywhere, i.e. cache all along the path from content store to the node with registered interest.

**LCD:** Leave Copy Down, i.e., bring the content down one step closer to interest [35].

**ProbCache:** Cache along the path from Interest to server probabilistically to accommodate multiple flows using this path [14].

**FixCache:** Cache along the path from Interest to server probabilistically with probability 0.1.

For these above-mentioned schemes, we used the *shortest path routing (SPR)* based interest dissemination towards the repository. We also compare this with two *nearest replica routing (NRR)* based forwarding, denoted by NRR' and NRR" that are proposed in [36]. NRR' uses an exploration phase where a content request is flooded by any node that receives the request. When a node finds the matching chunk in its cache, it forwards it towards the requested node. This may result in multiple chunks in return, and thus stress the network. NRR" runs in two phases: in the first phase the requesting node sends a meta-interest packet, indicating that it expects a binary reply regarding a content's availability. Based on the replies of the first phase, the requesting node sends an interest packet to the nearest node having the available content. NRR" ensures low traffic load and avoids cache pollution; however, it introduces delay due to the two phase exploration. In [36] the authors have shown that the nearest replica routing along with LCD shows superior performance, thus, we use NRR+LCD as a comparison benchmark.

We assume that the popularity distribution of the contents is Zipf with decay parameter $\alpha$. Note that $\alpha = 0$ implies a uniform distribution, and a larger $\alpha$ implies a distribution with shorter tail. Unless explicitly mentioned, the entire cache space is divided among the STC and LTC, with a ratio of 1:3. We assume a total content pool of $10^4$ with identical content-chunks for most of the simulations, whereas larger number of contents are studied in section VI-G. As we have assumed

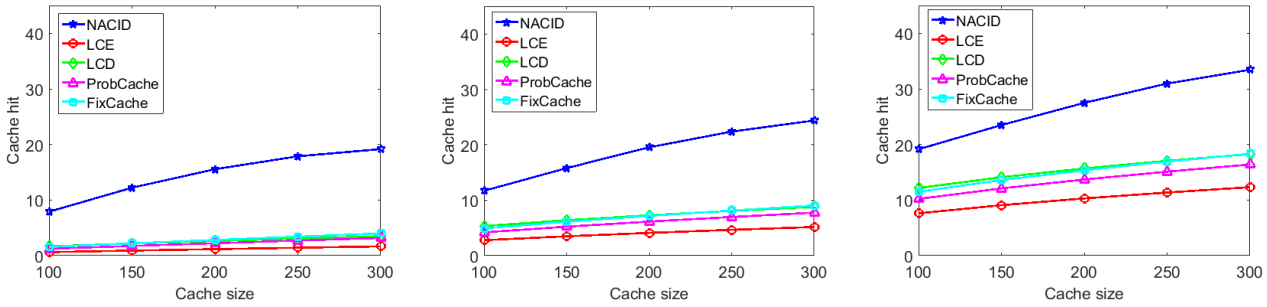



Fig. 11. Comparison of cache hit ratios for different caching schemes, with (a) $\alpha$ = 0.5, (b) $\alpha$ = 0.8, (c) $\alpha$ = 1.

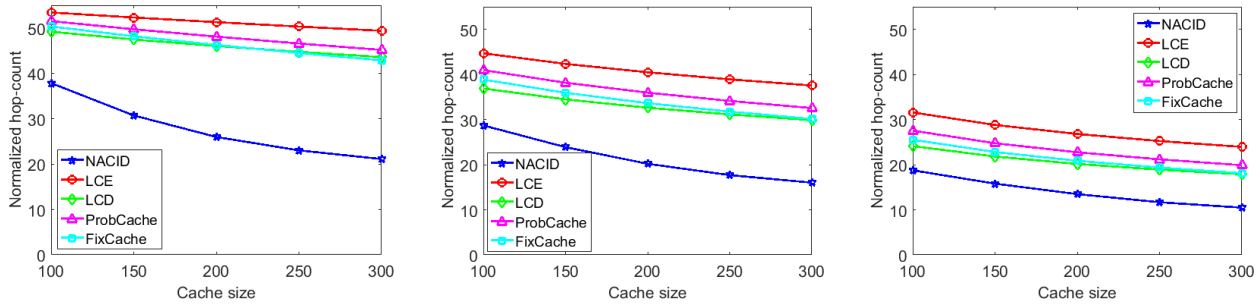



Fig. 12. Comparison of normalized hop-counts for different caching schemes, with (a) $\alpha$ = 0.5, (b) $\alpha$ = 0.8, (c) $\alpha$ = 1.

identical content-chunks, the cache sizes are defined by the number of chunks that the cache can accommodate. We use $\alpha$ =0.5, 0.8, and 1.0 for the results.

### A. Performance comparison with other schemes

For our simulations, the size of the bloom filter is determined as follows. In a BF, the presence of collision regions generate positive matches for a membership check of a content that is actually not present inside the set. A larger BF yields smaller false positive rate. Let $M$ denote the number items that may be inserted using the BF. The false positive probability $p$ is minimized if the length of the BF is optimally chosen to be $m = (-M \ln p)/(\ln 2)^2$ [37]. The corresponding optimal number of hash functions to be used is equal to $k = (m \ln 2)/M$. For our simulations, we choose the maximum cache size to be 500; with these the values of $m$ and $k$ are chosen to be ∼900 bytes and 10 respectively to keep $p$ approximately 0.001. Our current implementation of Bloom filter is borrowed from [38], which uses CRC32-128 bit hash to generate the hash values.

We compare the *Cache Hit Percentage* and the *Normalized Hop-Count* (NHC) for the above schemes. The former represents the probability that an Interest message finds the chunk in a cache, and the latter gives the percentage of the network diameter the Interest must walk before getting to the chunk.

*Performance of cache hit percentage:* Fig. 11 shows the cache hit percentage of NACID in comparison to other schemes, with the variation of cache sizes. From Fig. 11(a) we can observe that with $\alpha$ = 0.5, NACID improves the cache hit probability upto ∼5 times compared to the other schemes. With higher $\alpha$ (i.e. $\alpha$ = 1), this improvement becomes upto ∼2.75 times compared to other schemes. This is because for large $\alpha$, most of the popular contents are stored in the cache and at the same time accessed more frequently, which makes other schemes perform close to NACID. This shows the effect of caching the chunks based on their overall benefit, rather than some implicit information or some probabilistic inference.

We can also observe that the hit probability increases by upto 2.4 times, when the cache size increases from 100 to 300 based on different $\alpha$. This is obvious because more cache size accommodates more chunks, which improves the number of hits. We can also observe that the hit probability almost doubles when the $\alpha$ increases from 0.5 to 1. This is because with the increase in $\alpha$, more popular chunks are fetched more often, which overall improves the cache hit probability.

*Performance of normalized hop-counts:* Fig. 12 shows the comparison of the normalized hop-counts of NACID against other proposals. With $\alpha$ = 0.5, NACID reduces the number of hops traversed by ∼24%-57% compared to others. Similar improvements are also evident with higher $\alpha$. This clearly shows the improvement of NACID due its neighborhood awareness. We can also observe that the NHC goes down by ∼43% when the cache size is varied from 100 to 300. This is obvious because of the fact that higher cache size increases the number of cache hits and effectively improves the number of hops traversed. With the increase in $\alpha$ from 0.5 to 1, the NHC reduces by ∼33-56% since higher $\alpha$ concentrates most accesses to fewer chunks.

The results show that *the NACID algorithm improves the cache hit ratio upto 5 times over all other algorithms, and it does so while also simultaneously reducing the NHC by upto 57%*. This establishes the superiority of our algorithm over previous CCN caching algorithms, with only a small increase

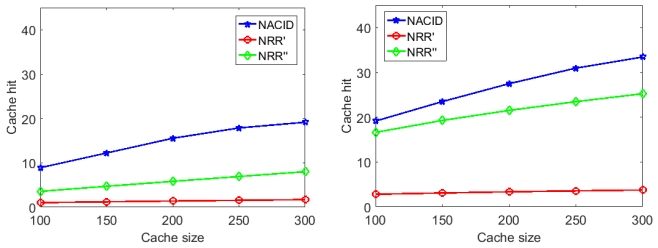



Fig. 13. Comparison of cache hit ratios for NACID, NRR', NRR", with (a) $\alpha$ = 0.5, (b) $\alpha$ = 1.

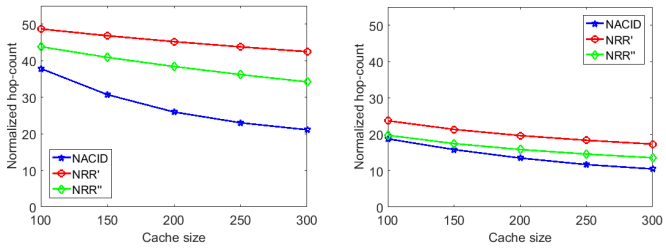



Fig. 14. Comparison of normalized hop-counts for NACID, NRR', NRR", with (a) $\alpha$ = 0.5, (b) $\alpha$ = 1.

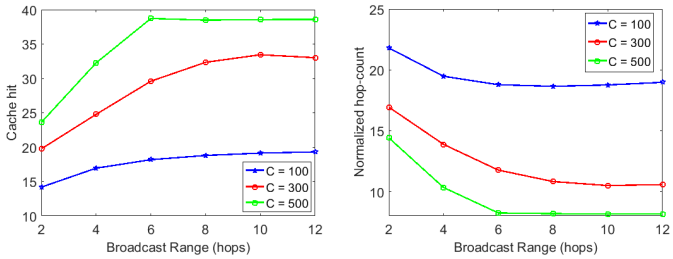



Fig. 15. Comparison of (a) cache hit ratios and (b) normalized hop-counts with broadcast range.

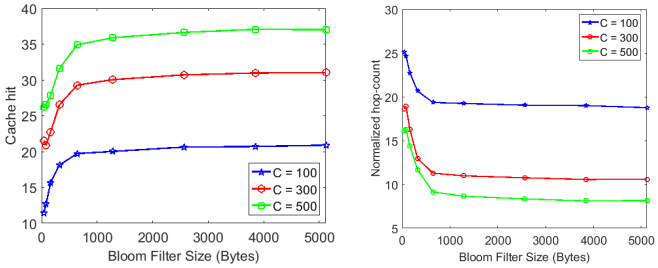



Fig. 16. Comparison of (a) cache hit ratios and (b) normalized hop-counts with different bloom filter sizes.

in the complexity. Among the other schemes, LCE performs worse than others because it always cache the contents along the path from the content store to the source of the interest.

*Comparison with NRR' and NRR":* Fig. 13-14 show the comparison of NACID with NRR' and NRR" schemes. From Fig. 13 we can observe that NACID increases the cache hit by upto 10 times as compared to NRR' and more than twice in case of NRR", with $\alpha$ = 0.5. This is because NACID not only exploits the advantage of neighborhood aware interest dissemination, but also takes advantage of caching the popular chunks by using simple SMA based popularity prediction along with their neighborhood aware collaborative benefit calculation. We can also observe that the improvement between NACID and NRR" decreases when $\alpha$ = 1. This shows that the popularity prediction gives more advantage with lower $\alpha$. This is intuitive because with higher $\alpha$, the content popularity becomes more skewed, thus, the gain from the collaborative benefit calculation reduces. Because of these advantages, NACID reduces the normalized hop-counts by ∼22-50% as compared to NRR' and ∼13-38% as compared to NRR", with $\alpha$ = 0.5, while the improvement decreases with higher $\alpha$.

Fig. 13-14 show an interesting observation, which is the performance penalty of NRR' due to its request interest flooding from the requesting node and the multiple cache evictions due to the return of multiple chunks in response to this flooding. While comparing in between NRR' and NRR" we can observe that with LCD, NRR" improves the cache hit by more than 3 times with $\alpha$ = 0.5. While the performance of NRR" improves with higher $\alpha$, NRR' still shows poor performance. Because of this reason with $\alpha$ = 1, NRR" improves the cache hit by upto 7 times as compared to NRR'. This penalty also results in higher hop-counts for NRR' as seen from Fig. 14.

### B. Performance of NACID with different tuning parameters

*1) Comparison with different broadcast range:* Fig. 15 shows the variation of cache hit ratio and normalized hop-traversal with different broadcast ranges, when $\alpha$ is assumed to be 1. From Fig. 15 we can observe that the cache hit ratio improves by ∼35-65%, and the NHC reduces by ∼14-43% when $\mathcal{B}$ increases from 2 to 12. This is because with more broadcast range, the content routers become more informed about the LTC contents around their neighborhood, which improves the network performance. However, this improvement comes at the cost of more control overhead. Notice that the improvement becomes marginal beyond $\mathcal{B}$ = 6 hops. Thus most of the contents are available within 6 hops around a router's neighborhood.

*2) Comparison with different BF size:* The bloom filter size plays a significant role in NACID performance due to its false positive effects. Fig. 16 shows the effects of bloom filter size on the cache hit ratio and NHC, where the $\alpha$ is assumed to be 1. From Fig. 16 we can observe that the hit ratio increases by ∼42-82%, whereas the NHC reduces by ∼28-50% when the filter size is increased from 40 to 5120 bytes. This is due to the false positive effects of the BF especially when the size is small. Due to the false positive effects, some interest packets are forwarded to wrong routers which leads to lower cache hit and higher NHC. However, beyond 640-1280 bytes the improvement starts saturating, as beyond that the false positive effects are marginal.

### C. Comparison with real datasets:

We next compare NACID with others using several real datasets including `Kosarak` and `Retail` that can be considered to follow the power law approximately. These datasets

| | Kosarak | Retail | Q148 | Nasa | T10I4D100K | T40I10D100K | Chess | Connect | Mushroom | Pumsb | Pumsb_star | Accidents |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Count** | 8019015 | 908576 | 234954 | 284170 | 1010228 | 3960507 | 118252 | 2904951 | 186852 | 3629404 | 2475947 | 11500870 |
| **Distinct items** | 41270 | 16470 | 11824 | 2116 | 870 | 942 | 75 | 129 | 119 | 2113 | 2088 | 468 |
| **Min** | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| **Max** | 41270 | 16469 | 149464496 | 28474 | 999 | 999 | 75 | 129 | 119 | 7116 | 7116 | 468 |
| $\alpha$ | 1.9979 | 1.5533 | 1.1104 | 2.0735 | 0.9906 | 0.9751 | 1.0865 | 1.7260 | 1.6361 | 2.4399 | 2.3389 | 3.7787 |

Fig. 17. Statistical characteristics of the datasets used.

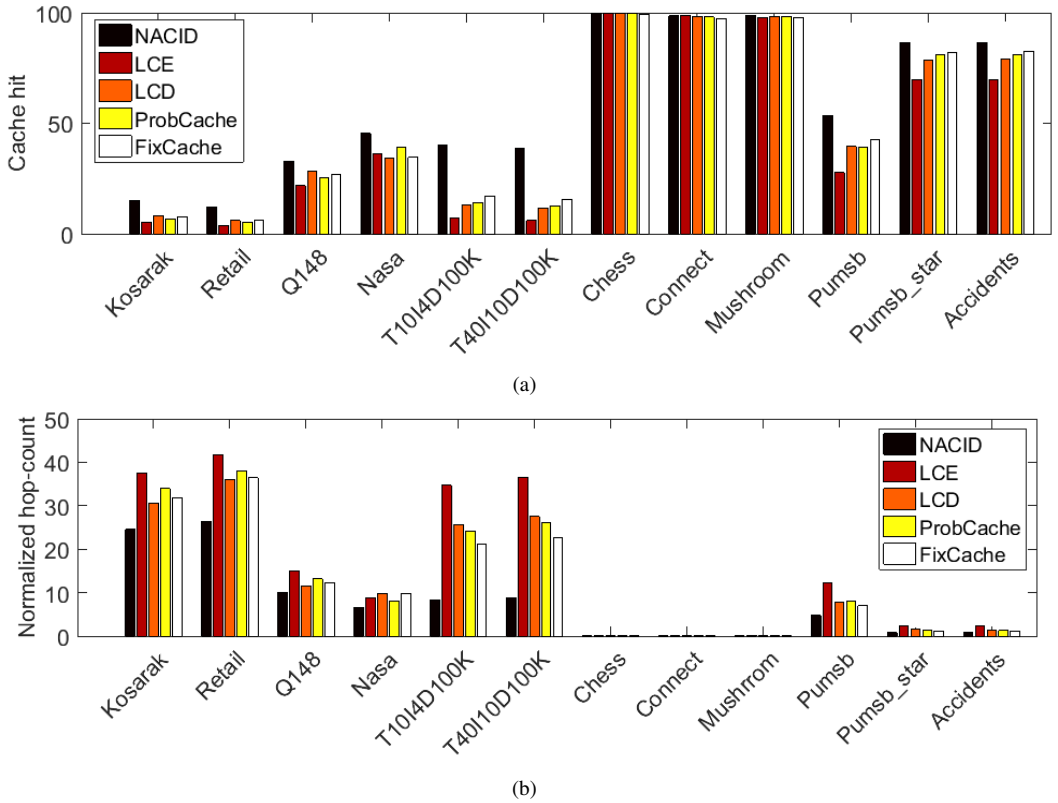



Fig. 18. Comparison of (a) cache hit ratios and (b) normalized hop-counts corresponding to different real datasets.

are publicly available and are widely used in data mining literature. We use twelve datasets that have diverse characteristics, as shown in Fig. 17. These are described in the following:

**Q148:** This dataset is derived from KDD Cup 2000 data, compliments of Blue Martini.

**Nasa:** This dataset is derived from the "Field Magnitude" and "Field Modulus" attributes from the Voyager 2 spacecraft Hourly Average Interplanetary Magnetic Field Data and the Voyager 2 Triaxial Fluxgate Magnetometer principal investigator, from NASA.

**IBM Almaden dataset:** The datasets `T10I4D100K` and `T40I10D100K` are generated using the generator from the IBM Almaden Quest research group.

**UCI/PUMSB datasets:** The datasets `chess`, `connect`, `mushroom`, `pumsb`, `pumsb_star` are prepared by Roberto Bayardo from the UCI datasets and PUMSB. `Chess` and `Connect` are gathered from game state information and are available from the UCI Machine Learning Repository [22], [39]. `Pumsb` and `Pumsb_star` datasets contain population and housing related census data.

**Accidents:** This dataset is donated by Karolien Geurts and contains anonymized traffic accident data [22], [40].

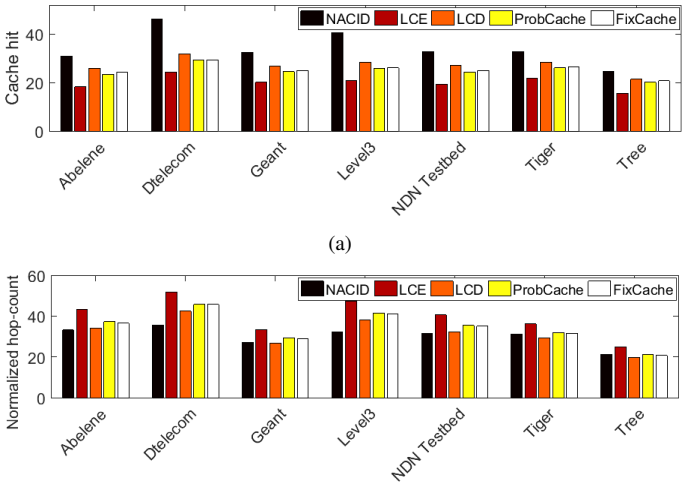
Although several of these datasets are not obtained in the CCN context (like `Mushroom`, `Accidents` etc.), they show approximately Zipf distribution and thus can be studied *as a representative datasets for content popularity distribution* as seen in Fig. 18.

We divided the contents obtained from these datasets among individual content routers, and considered them as their content requests. We assume the cache size to be 100. Fig. 18 shows the performance of NACID compared to the other schemes. For `Kosarak`, `Retail`, `T10I4D100K` and `T40I10D100K` datasets, NACID improves the cache hit by ∼2-3 times, whereas the hop-count is reduced upto 2-3 times. NACID also shows 22%-57% improvement in terms of cache hit and ∼13%-33% improvement in NHC with `Q148` dataset, compared to the other schemes. For `Nasa` dataset, the improvement of cache hit and NHR are ∼15%-32% and ∼23%-33% respectively. The performances are similar for `Chess`, `Connect` and `Mushroom` datasets. For the other datasets (i.e. `Pumsb`, `Pumsb_star` and `Accidents`), NACID improves the cache hit and NHC by ∼6%-96% and ∼1.5 times respectively.

The hit ratio of `Chess`, `Connect` and `Mushroom` are much higher compared to the other two datasets, because of fewer distinct contents in these datasets. For a similar reason,

| | $\|V\|$ | $\|E\|/\|V\|$ | $CoV$ | $D$ |
|---|---|---|---|---|
| **Abilene** | 11 | 2.5455 | 0.2052 | 5 |
| **DTelecom** | 68 | 10.3824 | 1.2917 | 3 |
| **Geant** | 22 | 3.3636 | 0.4159 | 6 |
| **Level3** | 46 | 11.6522 | 0.8739 | 4 |
| **NDN Testbed** | 17 | 3.7647 | 0.4150 | 5 |
| **Tiger** | 22 | 3.6364 | 0.1809 | 5 |
| **Tree** | 127 | 1.9843 | 0.5039 | 12 |
| **Grid100** | 100 | 3.6 | 0.1579 | 18 |

Fig. 19. Statistical characteristics of the network topologies.




(a)

(b)

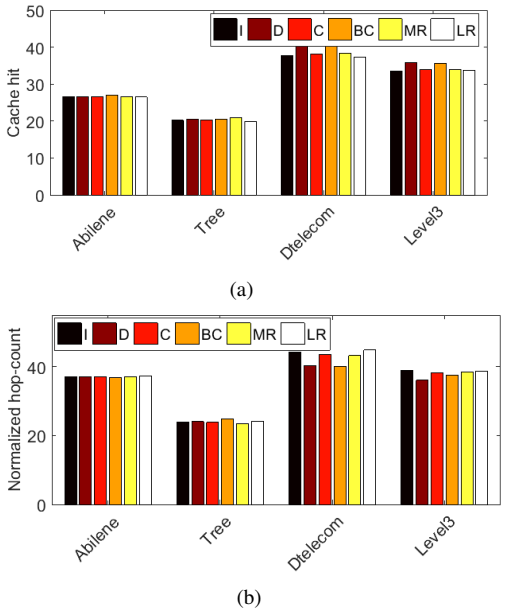
Fig. 20. Comparison of (a) cache hit ratios and (b) normalized hop-counts corresponding to different network topologies.

the NHR is also small for these datasets. Among the others, `Pumsb_star` and `Accidents` perform significantly better, mainly because of fewer distinct contents and/or higher $\alpha$.

### D. Comparison with different network topologies

We next show the comparison of NACID with others for different network topologies. To cover different types of networks, we consider both sparse (Abilene, Geant, NDN Testbed, Tiger, Tree) and dense (Dtelecom, Level3) network topologies. Fig. 19 shows the key characteristics of each graph, namely, the network size $|V|$, the average degree $|E|/|V|$, the coefficient of variation of the node degree $CoV$, and the graph diameter $D$.

From Fig. 20 we can observe that NACID improves the hit ratios by $\sim$20%-92% and decreases the NHC by up to $\sim$31% compared to the other schemes. We can observe that the improvement is maximum in case of Dtelecom and Level3 topologies because of their higher node degree (i.e. $|E|/|V|$) compared to the other network topologies. This is because a dense network provides NACID higher chances of fetching the content from the neighborhood caches corresponding to a particular router. Also in a dense network a neighborhood aware caching strategy can intelligently place the content-chunks among the neighborhood caches, so that the chunks are mostly available in a router's neighborhood if not in its local cache.




(a)

(b)

Fig. 21. Comparison of (a) cache hit ratios and (b) normalized hop-counts in case of heterogeneous caching.

### E. Homogeneous Caching vs Heterogeneous Caching

Next we show the effect of NACID in presence of heterogeneous cache size of the routers, where we assume that a total amount of cache memory is distributed among the content routers. We distribute the cache memory by analyzing the level of centrality of the routers within a network. As the central routers of a network serve more content requests, they are assigned more cache space as explained later on. We consider the following centrality metrics for this purpose:

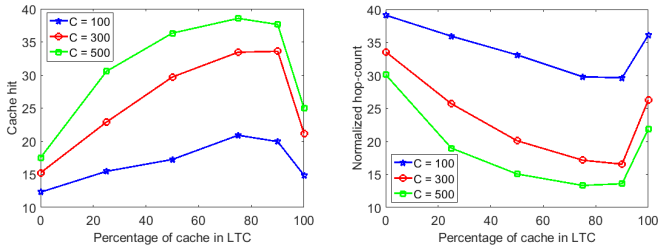
**Degree centrality (D):** Degree centrality of a router is defined as the number of links incident on that router, or the node degree of the routers.

**Closeness centrality (C):** Closeness centrality of a router is calculated as the sum of the length of the shortest paths between the router and all other routers in the network. Thus the more central a router is, the closer it is to all other routers. Thus closeness centrality of a router $x$ is given by $c(x) = \frac{1}{\sum_y d(y,x)}$[2], where $d(y,x)$ is the distance between $x$ and $y$.

**Betweenness centrality (BC):** Betweenness centrality of a router is the fraction of all shortest paths in the network that contain a given router. Routers with high values of betweenness centrality participate in a large number of shortest paths. Thus, betweenness centrality of a router $x$ is given by $g(x) = \sum_{s \neq x \neq t} \frac{\sigma_{st}(x)}{\sigma_{st}}$, where $\sigma_{st}$ is the total number of shortest paths from $s$ and $t$, and $\sigma_{st}(x)$ is the number of those paths that pass through $x$.

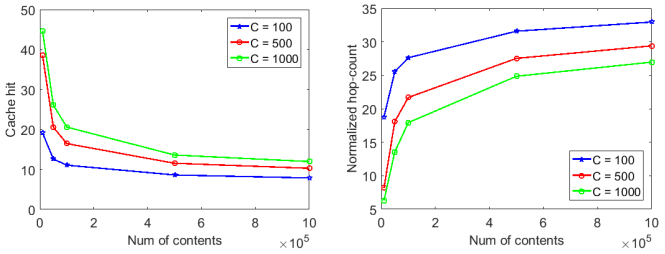
**More cache close to Repo (MR):** We also adopt a cache deployment strategy where the cache sizes of the routers that are closer to the Repo are more than that of the routers that are farther away. The intuition is that the routers that are closer to the Repo serve more requests, and thus putting larger caches there will be beneficial. We thus devise a metric, named Repo-

[2]For a large network, if $c(x)$ is very small, we can normalize it by multiplying this with the number of nodes or network diameter.




(a) (b)

Fig. 22. Comparison of (a) cache hit ratios and (b) normalized hop-counts with the percentage of cache in LTC.




(a) (b)

Fig. 23. Comparison of (a) cache hit ratios and (b) normalized hop-counts with different number of contents.

closeness of a router $x$ which is given by $r(x) = 1/D_x$ where $D_x$ is the shortest distance from router $x$ to the Repo.

**Less cache close to Repo (LR):** We next consider the scenario where larger caches are assigned to the routers that are far away from the Repo. The intuition behind this is that the interest packets from the far-away routers need to traverse more hops to reach the Repo, and so they are assigned larger caches. We thus develop a metric $l(x) = D(x)$ to model the LR cache distribution.

We assume that the total cache size of the topology is fixed and is assumed to be $C_{\text{tot}}$. In case of homogeneous caching (or identical caching **I**), we divide the cache equally among the routers, i.e. $C_i = C_{\text{tot}}/|V|$. However, in case of heterogeneous caching the cache space of router $i$ is given by

$$C_i = \left\lceil \frac{X_i}{\sum_{j \in V} X_j} C_{\text{tot}} \right\rceil \tag{6}$$

where $X_i$ is the suitable metric for router $i$ depending on which caching strategy (**D**, **C**, **BC**, **MR**, **LR**) is adopted.

Fig. 21 shows the performance of NACID with heterogeneous caching schemes. We assume $\alpha$ = 1 for this set of figures, and $C_{\text{tot}}$ is assumed to be $100\times|V|$. From this figure we can observe that the performance of NACID does not change significantly with heterogeneous cache distribution. Only a modest performance gain of <1.15x (in cache hit) is observed in case of Dtelecom and Level3 compared to its homogeneous counterpart. Similar findings are also observed in [41]. This leads to the conclusion that there is no real incentives of using heterogeneous caching as opposed to homogeneous caching strategy in case of NACID.

### *F. Comparison with the distribution of cache across STC and LTC*

Fig. 22 shows the how the cache hit and NHC change with different cache distribution across STC and LTC, with $\alpha = 1$. From Fig. 22 we can observe that when we increase the storage in the LTC, the cache hit starts increasing (whereas the NHC starts decreasing) till 70–90% of the storage is assigned to LTC, and then starts dropping. This phenomenon can be explained as follows. When STC:LTC = 100:0, all the cache storage is assigned to the STC. Thus no contents go to the LTC, and therefore no LTC cache information sharing can take place among the neighbors of the content routers. Because of this reason, the benefits of neighborhood awareness of NACID becomes ineffective, which hurts the performance. With the increase in LTC storage, the scheme takes advantage of the neighborhood awareness, which improves the overall performance. However, the size of the STC starts shrinking, thus there is less room for the new-coming contents to be cached. When STC:LTC = 0:100, no new contents are cached after the cache is filled up. However, even in this situation the performance is better than that in case of LTC = 0, because of its neighborhood awareness.

TABLE I
MEMORY OVERHEAD FOR STORING BLOOM FILTER

| Cache size | False positive | BF size | Memory ($n$ = 20) | Memory ($n$ = 50) |
|---|---|---|---|---|
| $10^5$ | $10^{-3}$ | 175.51 KB | 3.5 MB | 8.7 MB |
| | $10^{-6}$ | 351.02 KB | 7.02 MB | 17.55 MB |
| | $10^{-9}$ | 526.52 MB | 10.5 MB | 26.32 MB |
| $5\times10^5$ | $10^{-3}$ | 877.54 KB | 17.5 MB | 43.87 GB |
| | $10^{-6}$ | 1.71 MB | 34.2 MB | 85.5 MB |
| | $10^{-9}$ | 2.57 MB | 51.4 MB | 128.5 MB |
| $10^6$ | $10^{-3}$ | 1.71 MB | 34.2 MB | 85.5 MB |
| | $10^{-6}$ | 3.43 MB | 68.6 MB | 171.5 MB |
| | $10^{-9}$ | 5.14 MB | 102.8 MB | 256 MB |

### *G. Comparison with different number of Items*

Fig. 23 shows the performance of NACID where the number of items, say $N$, varies from $10^4$ to $10^6$. As discussed in section IV, $10^6$ items is quite adequate to study the performance of a regional router. We assume $\alpha$ to be 1 for this figure. To consider the stressed scenario, we made the lowest $\frac{C}{N}$ to as low as 0.01% in Fig. 23. In that case also NACID achieves a hit ratio of ∼8%. From Fig. 23 we can observe that the performance of NACID deteriorates with the increase in number of items. With $C$ = 1000, the hit ratio reduces from ∼45% to ∼12%, when $N$ is increased from $10^4$ to $10^6$; the corresponding NHC also increases by ∼3-4x. With the increase in cache size from 100 to 1000, the hit ratio increases by up to 2 times, whereas the NHC decreases by up to 3 times.

### *H. Overhead analysis for using Bloom Filter based content dissemination*

*1) Memory overhead:* We next estimate the memory overhead for disseminating and storing the bloom filters of all the neighbors in a router's Neighbor base. To consider a stress situation, we consider a rather small chunk size of 10KB and a rather large cache size of 10 GB. Reference [42]

argues that chunk size of less than 10KB is undesirable due to high overhead of headers and chunk management. Similarly, articles [43], [27] have studied the cache size issue and consider more than 10GB cache unrealistic. This results in a maximum number of chunks within a cache to be $10^6$, which we consider as a realistic upper limit of cached contents. Table I shows the additional memory requirement for storing the bloom filter in the Neighbor base with different number of neighbors (assumed to be $n$) and false positive rates. From Table I we can observe that even with $C$ = $10^6$ and $n$ = 50, the additional memory requirement is 256 MB which is $<$3% of the cache memory of 10 GB. This shows that the additional overhead of storing the bloom filters are relatively small and thus can be integrated into real CCN environments.

*2) Computational overhead:* For every incoming request a router first needs to do a look-up in the local caches (STC and LTC). This can be implemented by an hash-table. The requested chunk-id is hashed and the entry at the hash address is searched for a match. With good hashing and suitable hash table size, this look-up operation takes $\mathcal{O}(1)$ amount of time. When a chunk is cached, its chunk-id is hashed and its cache address is stored in the hash table. This operation also takes a constant time. The deletion operation from the hash-table also takes a constant time.

If the chunk corresponding to a *local* request is found in the cache, then it is served. Otherwise the router (say $x$) originates an Interest packet. The router needs to do a look-up in its neighbor's BF from its Neighbor Base, each at a constant cost. If the chunk is not found in the Neighbor Base (based on the BF look-up) then the Interest packet is sent to the Repo. Otherwise it is sent to the neighboring router (say $y$) with the least cost. This entire operation takes $\mathcal{O}(n)$ time, where $n$ is the number of entries the Neighbor Base of $x$. The intermediate routers in between $x$ and $y$ (or Repo) check whether the chunk exists in their local caches or not in $\mathcal{O}(1)$ time. If the chunk is found at $y$ it is served, otherwise it is sent to the Repo. All these operations take constant time.

When a chunk arrives at a router, it takes the caching decision/replacement (if any) for STC. This is done using the Heap mechanism discussed in section V-A. The checking only takes constant time, but the subsequent heap reorganization (not on the critical path) takes $\mathcal{O}(\log m)$ time (where $m$ is the number of entries in STC). As the STC size will typically 20-25% of the cache size (which is $10^5 - 10^6$ as obtained from Table I), this operation is pretty fast.

Now let us consider the computation of the benefit ($w_i$ in equation (4)). The popularity predictions of the chunks are done infrequently (in few minutes to hours) as typically the popularity of the Internet contents do not change too frequently. Ideally the recency of the contents (i.e. $\Delta_i$) need to be updated continuously. However in case of a realistic implementation, this factor can be updated in an infrequent fashion (in few minutes). Similarly the reshuffling of the contents within the LTC and the STC depending on their benefit functions is also infrequent.

## VII. Results from Real Video Traces

We now show the performance of NACID on Youtube video traces collected from a gateway router of University of Massachusetts, Amherst [19]. The traces are collected by monitoring the traffic between clients in the campus network and YouTube servers. Trace-1 was collected for 3 days in the last week of Spring 2007 semester, and consists of a total of 32367 requests. Trace-2 consists of the requests generated during the two weeks at the beginning of the Spring 2008 semester, whereas Trace-3 consists of seven consecutive 24 hours traces during the Spring 2008 semester. The overall statistics of these three traces are enlisted in Table II. Fig. 24(a)-(c) shows the number of times the contents has requested versus their ordering, which again shows the Zipf distribution with the exponents equal to 0.3842, 0.5959 and 0.5198 respectively.

With these traces we have considered the $10\times10$ grid topology, divided the contents from these traces among individual the routers, and considered them as their content requests. Fig. 24(d)-(e) show the result of cache hit ratio and NHC of these three traces with different cache sizes. From these figures we can observe that NACID achieves a cache hit of $\sim$15% with $C$ = 500 in case of `Trace-1`, which results in a NHC of 40. On the other hand larger traces like `Trace-2` and `Trace-3` take a cache size of 4000 and 2000 respectively to reach a cache hit of $\sim$9-10%. This is because these two traces consist of a significant number of distinct contents, with a content size of $\sim$303K and $\sim$141K respectively, which leads to a $\frac{C}{N}$ ratio of less than 1.5% with the cache size of 4000 and 2000 respectively. Also observe that for `Trace-2` and `Trace-3`, the cache hit and NHC start saturating after a cache size of 4000. This can be explained from the popularity distribution of Fig. 24(b)-(c), which shows that the number of accesses of the contents after the top 4000 are even less than 10, thus making the cache size even bigger will not result in further improved performance.

## VIII. Related Works

*Caching in General:* Cooperative caching has been studied extensively in different environments such as World wide web, peer-to-peer system, as well as in the file and storage systems. Cooperative web caching is explored including hierarchical, hash-based and directory-based caching schemes in [9]. Cache management in peer-to-peer storage system has been presented in [10], that replicates multiple copies of a file to reduce access latencies. Coordinated caching of multiple clients in a LAN is presented in [11] to improve the performance of a network file system.

However such caching schemes run at the end peers and proxies over an IP layer, whereas in CCN caching is targeted to be done in every router. At the same time in CCN caching management needs to be done at line-speed of the routers to make it universal. Caching in content distribution networks (CDN) are explored in [44], [45]. CDN is essentially an overlay infrastructure where caching is done only at the content distribution routers, which is different from CCN caching which is universal in nature.

TABLE II
YOUTUBE VIDEO TRACES [19]

| | Date | Duration | Count | Distinct items | $\alpha$ |
|---|---|---|---|---|---|
| **Trace-1** | 05/23/07 – 05/26/07 | 72 hours | 32367 | 23244 | 0.3842 |
| **Trace-2** | 01/29/08 – 02/12/08 | 336 hours | 611968 | 303332 | 0.5959 |
| **Trace-3** | 03/11/08 – 02/18/08 | 168 hours | 243023 | 141177 | 0.5198 |

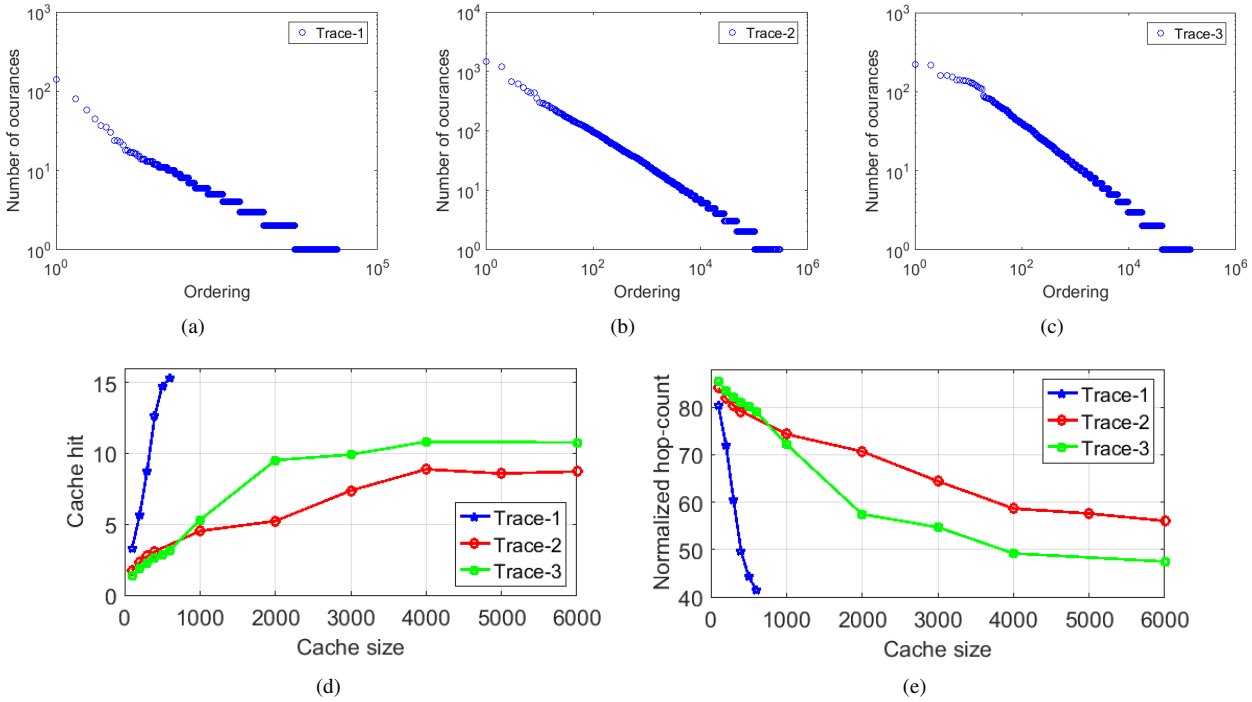



Fig. 24. (a)-(c)Frequency of content accesses versus content ranking for the Youtube traces obtained from UMass, Amherst dataset [19], along their corresponding (d) cache hit and (e) normalized hop counts.

*Caching Decision and Forwarding Policy in CCN:* Caching in CCN is also well researched topic, however, in most of the proposed schemes a content router does not need to know the cache information in its neighborhood, and the caching decisions are taken autonomously by the routers. Leave copy down (LCD) [35], Move copy down (MCD) [35], copy with some probability [35] and Probabilistic cache [14] falls in this category. In [12], the authors have argued that the chunks of a file is correlated or fetched in a sequential manner. They have proposed a scheme named WAVE, where the routers exponentially increase in the number of chunks cached for a file with the increase in the number of requests. In [46] the authors have proposed a label-based caching where the nodes cache a specific range of contents (defined as levels), defined a priori. A popularity based cache consistency mechanism is designed in [47]. In [48] the authors have discussed the optimal cache allocation problem in CCN with an objective of optimally distributing the cache capacity across the CCN routers under the constraint of total network storage budget. A number of surveys on content caching is reported in [49], [50], [51], whereas energy aware caching is studied in [52], [52].

For the forwarding of interest packets, shortest path routing towards the repository is typically used. In [53] the authors have proposed a hierarchical architecture where the consumers are at the lowest level, whereas the publishers are at the highest level. Upon arrival of a request at any layer, the request is forwarded randomly on any of the outgoing links within the same layer up to a certain time-threshold. When this threshold is exceeded, the request is forwarded to the next upper level. The nearest replica routing based forwarding is proposed in [36], [54], however, this requires flooding the request packets during the exploration phase, which causes extra traffic, delay, and unnecessary cache evictions. To limit the effects of flooding, in [55], [56] the authors propose flooding only for popular contents based on an exponentially fading probability. In [57] the authors have studied the effects of scoped-flooding for content discovery in an information-centric network. Stateful forwarding is proposed in [58] where the links are ranked to be either green, yellow or red. A link is considered as green if no delivery failure or congestion has been acknowledged, yellow when congestion has been acknowledged, and red when the transmission is unlikely or a link failure has occurred. Upon the arrival of a request, a node prefers the green paths over the yellow ones, whereas the red paths are avoided. In [59] the authors have used an Ephemeral Forwarding Information Base (EFIB) to keep track of the direction in which the data packets were temporarily cached in the recent past; the EFIB opportunistically creates an entry by a returning data packet so that a matching interest packet

can follow towards the direction, where the corresponding data has been recently cached.

In contrast, in NACID the CCN routers occasionally forward Bloom Filter to share their cached contents, so that the routers can (a) use this information while caching the content-chunks, and also (b) forward their content interest to their neighboring caches rather than always forwarding them towards the content store. This makes NACID unique in that it does not use network-wide flooding and the neighborhood aware caching yields lower hop count and higher cache hits.

*Cache Replacement Policy in CCN:* The most common cache replacement policy is Least Recently Used (LRU) policy where the least recently accessed content is replaced with a newly arriving content when the cache is full. However LRU captures the freshness of a content, but not its frequency of accesses. Some variants of LRU are LRU-K [60] and 2Q [61]. Least Frequently Used (LFU) is an alternative of LRU to capture the access frequency. ARC [62] captures both the recency and frequency of a cache's contents by keeping track of two lists: one keeps track of the recently accessed contents whereas other one records the frequently accessed contents.

*Content Popularity in CCN:* Content popularity distribution is studied extensively in [18], [19]. The studies conclude that the content popularity in CCN follows heavy-tailed distributions, which can be modeled as a power-law distribution. They also observe that a significant portion of the contents are just one-timers. The effect of short-term and long-term content popularity is studied in [63], [64]. The value of the scale-factor of the popularity distributions varies in the literature from 0.6 [20] to 2.5 [21].

*Bloom Filter:* The Bloom filter data structure was first introduced by Burton H. Bloom in 1970 [65]. Since then Bloom filter has been used in various domains including Web caching [66], P2P networks [67], packet routing and forwarding [68], RFID tag identification [69], differential file access in DBMS systems [70] etc. There are different variants of Bloom filters that are proposed in the literature, such as counting Bloom filter [71], compressed Bloom filter [72], deletable Bloom filter [73], hierarchical Bloom filter [74] etc. The use of Bloom filter in CCN has been studied in [75], [76], [77], [78].

## IX. Conclusions

In this paper, we investigated a neighborhood aware in-network cache management and information dissemination scheme in order to minimize content fetch latency in CCN. Three key features of NACID architecture are (a) the use of repositories for maintaining the content recency, (b) lightweight content information dissemination by the use of Bloom filters, and (c) using them for developing a neighborhood-aware two-level caching and interest forwarding scheme. The simulation results show that the NACID, compared to the existing caching algorithms, significantly increases hit ratio, and at the same time reduces the number of hops for fetching the contents. This performance improvement is consistent across different network topologies, as well as for different content mining datasets. We also consider the effects of various heterogeneous cache memory allocation strategies on NACID by using different graph centrality metrics. However, a thorough simulation comparison suggests that heterogeneous caching strategies can only affect the performance gain marginally and thus are insignificant in practice.

## References


[1] Cisco, "Cisco visual networking index: forecast and methodology, 2009-2014," Cisco, Tech. Rep., 2010. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[2] S. C. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *IEEE INFOCOM*, 2010, pp. 1478–1486.

[3] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Computer Networks*, vol. 57, no. 16, pp. 3128–3141, 2013.

[4] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[5] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, July 2012.

[6] M. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks," *IEEE Communications Magazine*, Dec 2012.

[7] J. Choi, J. Han, E. Cho, T. Kwon, and Y. Choi, "A survey on content-oriented networking for efficient content delivery," *IEEE Communications Magazine*, vol. 49, no. 3, pp. 121–127, 2011.

[8] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[9] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *ACM SOSP*, 1999, pp. 16–31.

[10] A. I. T. Rowstron and P. Druschel, "Storage management and caching in past, A large-scale, persistent peer-to-peer storage utility," in *ACM SOSP*, 2001, pp. 188–201.

[11] M. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching: Using remote client memory to improve file system performance," in *USENIX (OSDI)*, 1994, pp. 267–280.

[12] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *INFOCOM Workshops*, 2012, pp. 316–321.

[13] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networks." in *INFOCOM Workshops*, 2012, pp. 268–273.

[14] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *ACM ICN*, 2012, pp. 55–60.

[15] J. M. Wang and B. Bensaou, "Progressive caching in CCN," in *IEEE GLOBECOM*, 2012, pp. 2727–2732.

[16] A. Pal and K. Kant, "Iot-based sensing and communication infrastructure for the fresh food supply chain," *IEEE Computer*, Feb 2018.

[17] ——, "A food transportation framework for an efficient and worker-friendly fresh food physical internet," *MDPI Logistics*, Dec 2017.

[18] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: A view from the edge," in *IMC*, 2007, pp. 15–28.

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

[20] K. V. Katsaros, G. Xylomenos, and G. C. Polyzos, "Multicache: An overlay architecture for information-centric networking," *Computer Networks*, vol. 55, no. 4, pp. 936–947, 2011.

[21] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *ACM ICN*, 2011, pp. 26–31.

[22] "Frequent itemset mining dataset repository," http://fimi.ua.ac.be/data/.

[23] A. K. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks."

[24] R. Nau, "Forecasting with moving averages," https://people.duke.edu/ rnau/Notes_on_forecasting_with_moving_averages–Robert_Nau.pdf.

[25] R. Aufrichtig and S. B. Pedersen, "Order estimation and model verification in autoregressive modeling of eeg sleep recordings," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 1992, pp. 2653–2654.

[26] "Frequent items in streaming data: An experimental evaluation of the state-of-the-art," http://disi.unitn.it/ themis/frequentitems/.

[27] D. Perino and M. Varvello, "A reality check for content centric networking," in *ACM ICN*, 2011, pp. 44–49.

[28] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kâafar, and S. Uhlig, "Trace-driven analysis of ICN caching algorithms on video-on-demand workloads," in *ACM CoNEXT*, 2014, pp. 363–376.

[29] J. J. Garcia-Luna-Aceves, "Name-based content routing in information centric networks using distance information," in *ACM ICN*, 2014, pp. 7–16.

[30] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[31] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *IEEE INFOCOM*, 2016, pp. 1–9.

[32] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 604–612, 2002.

[33] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *ACM CoNEXT*, 2014, pp. 75–88.

[34] R. Chiocchetti, D. Rossi, and G. Rossini, "ccnsim: An highly scalable CCN simulator," in *IEEE ICC*, 2013, pp. 2309–2314.

[35] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Perform. Eval.*, vol. 63, no. 7, pp. 609–634, 2006.

[36] G. Rossini and D. Rossi, "Coupling caching and forwarding: benefits, analysis, and implementation," in *ACM ICN*, 2014, pp. 127–136.

[37] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[38] www.csee.usf.edu/ christen/tools/bloom2.c.

[39] D. Burdick, J. Gehrke, J. Flannick, T. Yiu, and M. Calimlim, "Mafia: A maximal frequent itemset algorithm," *IEEE Transactions on Knowledge & Data Engineering*, vol. 17, no. 11, pp. 1490–1504, 2005.

[40] K. Geurts, "Traffic accidents data set," http://fimi.ua.ac.be/data/accidents.pdf.

[41] D. Rossi and G. Rossini, "On sizing CCN content stores by exploiting topological information," in *IEEE INFOCOM*, 2012, pp. 280–285.

[42] G. Rossini, "Design analysis of forwarding strategies for host and content centric networking," https://perso.telecom-paristech.fr/drossi/paper/phd-thesis-giuseppe.pdf, 2014.

[43] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *Proceedings of the Re-Architecting the Internet Workshop*, 2010.

[44] K. Park and V. S. Pai, "Scale and performance in the coblitz large-file distribution service," in *NSDI*, 2006.

[45] M. J. Freedman, "Experiences with coralcdn: A five-year operational view," in *NSDI*, 2010, pp. 95–110.

[46] Z. Li and G. Simon, "Time-shifted TV in content centric networks: The case for cooperative in-network caching," in *IEEE ICC*, 2011, pp. 1–6.

[47] B. Feng, H. Zhou, H. Zhang, J. J. Jiang, and S. Yu, "A popularity-based cache consistency mechanism for information-centric networking," in *IEEE GLOBECOM*, 2016, pp. 1–6.

[48] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie, "Optimal cache allocation for content-centric networking," in *IEEE ICNP*, 2013, pp. 1–10.

[49] A. Ioannou and S. Weber, "A survey of caching policies and forwarding mechanisms in information-centric networking," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 4, pp. 2847–2886, 2016.

[50] I. Abdullahi, A. S. M. Arif, and S. Hassan, "Survey on caching approaches in information centric networking," *J. Network and Computer Applications*, vol. 56, pp. 48–59, 2015.

[51] I. U. Din, S. Hassan, M. K. Khan, M. Guizani, O. Ghazali, and A. Habbal, "Caching in information-centric networking: Strategies, challenges, and future research directions," *IEEE Communications Surveys Tutorials*, vol. 20, no. 2, pp. 1443–1474, 2018.

[52] C. Fang, F. R. Yu, T. Huang, J. Liu, and Y. Liu, "A survey of green information-centric networking: Research issues and challenges," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 3, pp. 1455–1472, 2015.

[53] G. de Melo Baptista Domingues, E. de Souza e Silva, R. M. M. Leão, D. S. Menasché, and D. Towsley, "Enabling opportunistic search and placement in cache networks," *Computer Networks*, vol. 119, pp. 17–34, 2017.

[54] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. M. Maggs, K. C. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: incrementally deployable ICN," in *ACM SIGCOMM*, 2013, pp. 147–158.

[55] R. Chiocchetti, D. Rossi, G. Rossini, G. Carofiglio, and D. Perino, "Exploit the known or explore the unknown?: hamlet-like doubts in ICN," in *ACM ICN*, 2012, pp. 7–12.

[56] R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, and G. Rossini, "INFORM: a dynamic interest forwarding mechanism for information centric networking," in *ACM ICN*, 2013, pp. 9–14.

[57] L. Wang, S. Bayhan, J. Ott, J. Kangasharju, and J. Crowcroft, "Understanding scoped-flooding for content discovery and caching in content networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1887–1900, 2018.

[58] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013.

[59] O. Ascigil, V. Sourlas, I. Psaras, and G. Pavlou, "A native content discovery mechanism for the information-centric networks," in *ACM ICN*, T. C. Schmidt and J. Seedorf, Eds., 2017, pp. 145–155.

[60] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," *SIGMOD Rec.*, vol. 22, no. 2, pp. 297–306, 1993.

[61] T. Johnson and D. Shasha, "2q: A low overhead high performance buffer management replacement algorithm," in *VLDB*, 1994, pp. 439–450.

[62] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache," in *FAST*, 2003, pp. 115–130.

[63] Y. Borghol, S. Mitra, S. Ardon, N. Carlsson, D. L. Eager, and A. Mahanti, "Characterizing and modelling popularity of user-generated videos," *Perform. Eval.*, vol. 68, no. 11, pp. 1037–1055, 2011.

[64] S. Mitra, M. Agrawal, A. Yadav, N. Carlsson, D. Eager, and A. Mahanti, "Characterizing web-based video sharing workloads," *ACM Trans. Web*, vol. 5, no. 2, pp. 8:1–8:27, 2011.

[65] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[66] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.

[67] H. Cai, P. Ge, and J. Wang, "Applications of bloom filters in peer-to-peer systems: Issues and questions," *IEEE NAS*, vol. 0, pp. 97–103, 2008.

[68] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: An aid to network processing," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 181–192, 2005.

[69] Y. Nohara and S. Inoue, "A secure and scalable identification for hash-based rfid systems using updatable pre-computation," in *ACM WiSec*, 2010, pp. 65–74.

[70] L. L. Gremillion, "Designing a bloom filter for differential file access," *Commun. ACM*, vol. 25, no. 9, pp. 600–604, 1982.

[71] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, 1990.

[72] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, 2002.

[73] C. E. Rothenberg, C. Macapuna, F. Verdi, and M. Magalhães, "The deletable bloom filter: a new member of the bloom family," *IEEE Communications Letters*, vol. 14, no. 6, pp. 557–559, 2010.

[74] K. Shanmugasundaram, H. Brönnimann, and N. Memon, "Payload attribution via hierarchical bloom filters," in *ACM CCS*, 2004, pp. 31–41.

[75] W. Quan, C. Xu, J. Guan, H. Zhang, and L. A. Grieco, "Scalable name lookup with adaptive prefix bloom filter for named data networking," *IEEE Communications Letters*, vol. 18, no. 1, pp. 102–105, 2014.

[76] W. Quan, C. Xu, A. V. Vasilakos, J. Guan, H. Zhang, and L. A. Grieco, "TB2F: tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking," in *IFIP Networking*, 2014, pp. 1–9.

[77] C. Tsilopoulos, G. Xylomenos, and Y. Thomas, "Reducing forwarding state in content-centric networks with semi-stateless forwarding," in *IEEE INFOCOM*, 2014, pp. 2067–2075.

[78] Y. Wang, K. Lee, B. Venkataraman, R. L. Shamanna, I. Rhee, and S. Yang, "Advertising cached contents in the control plane: Necessity and feasibility," in *IEEE INFOCOM*, 2012, pp. 286–291.