NACID: A Neighborhood Aware Caching and Interest Dissemination in 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, by exploiting ubiquitous in-network caching to minimize content delivery latency and reducing the 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 there 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 provides an increase in up to ∼3 times of cache hits, and decrease in up to ∼30% the number of hops required to get the contents than existing CCN caching policies.

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

I. Introduction

The tremendous growth of Internet traffic in recent past propels the necessity of modifying the Internet architecture in an efficient way. 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 VoD or time-shift TV services [2].

Most of these traffic are content retrieval applications. This compels the Internet designers to shift from 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 information availability, demands, and delivery needs (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.

This research was supported by the NSF grant CNS-1542839.

In-network content caching is not new and has been studied in other network domains such as Web service, P2P, CDN [9], [10], [11]. However such mechanism is not suitable for directly applied in 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, all of the schemes that we are aware of 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 a *Neighborhood Aware Caching and Interest Dissemination (NACID)* scheme where the caching decision is made based on whether there is 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 number of hops from the requesting node; however, more sophisticated metrics such as a given delay limit can also be considered. Also in a CCN different links have separate costs (capacities, traffic volumes, delay etc.), thus in NACID the nodes need to consider the routes in between 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 dissemination mechanism of caching information in the neighborhood so that its overhead remains small. We propose a Bloom Filter based light-weight cache dissemination approach for doing this. Given these, we next propose a *two-*

978-1-5090-2991-4/17/$31.00 ©2017 IEEE
Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.

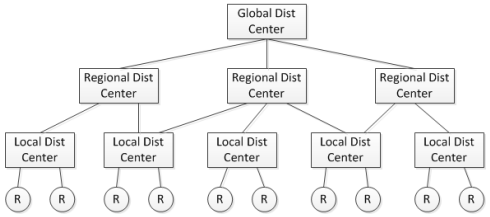


Fig. 1: An abstract model of PCDN. "R" denotes a retailer.

level caching architecture in NACID. In one level the caching decisions are taken in a *synchronous* (i.e., driven by the arrivals of the newer contents) manner, whereas *asynchronous* (i.e., done periodically as a housekeeping activity) policy is adopted for the other level. 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 $\sim$3 times, whereas the hop-reduction percentage goes down up to $\sim$30%.

The outline of this paper is as follows. Section II describes the key motivation behind developing the NACID architecture. Section III proposes the system model, content popularity distribution and network architecture assumed in our scheme. Section IV introduces the proposed neighborhood aware caching scheme and describes the operation and interaction of the two-level caching mechanism. Section V shows the simulation comparison of NACID against other well-known existing schemes. Relevant literature and discussions are summarized in section VI. Finally, the paper is concluded in section VII.

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], [18], [19]. 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.

At the same time 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 the similar characteristics in that the content can be fetched from some neighboring cache, rather than bringing from the actual source server as in IP.

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 needs 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 and section IV.

III. The System Model

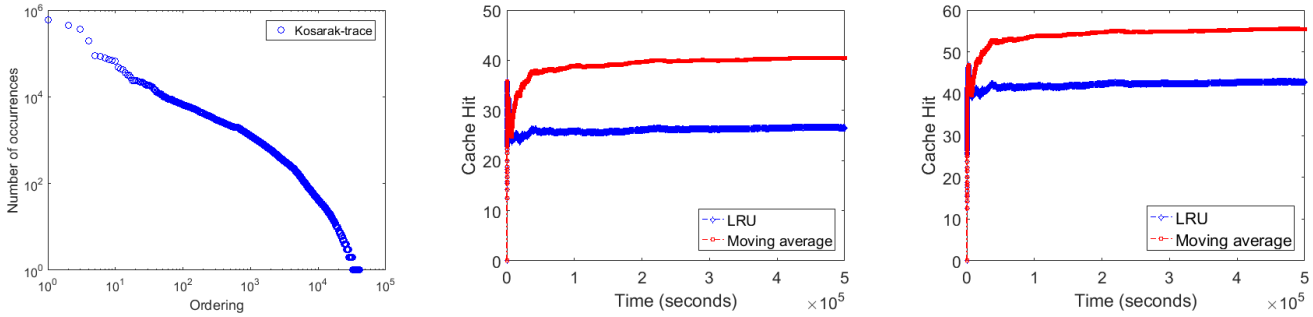
In CCN the content popularity is determined by how often a piece of content is requested. Recent studies [20], [21] show that the users are attracted to only few contents, while others are accessed rarely. In fact a significant portion of the contents are one-timers. Therefore, the content popularity is commonly modeled with the Zipf distribution function. In a Zipf distribution, out of the population of N contents, the frequency of the i-th content is given by

$$f(i,\alpha,N) = \frac{\frac{1}{i^\alpha}}{\sum_{j=1}^{N}\frac{1}{j^\alpha}} = \frac{\frac{1}{i^\alpha}}{H_{N,\alpha}} \tag{1}$$

where α is the Zipf exponent and $H_{n,\alpha} = \sum_{j=1}^{N}\frac{1}{j^\alpha}$ the generalized harmonic number of order α. In literature the range of α is varied from 0.6 [22] to 2.5 [23].

However, 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 varies spatially and temporally. To predict this dynamic and regional popularity, we assume a simple content demand prediction model using the *simple moving average model (SMA)* [25]. SMA is used for predicting time series, where the value of Y at time $t+1$

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.



(a) (b) (c)

Fig. 2: (a) Frequency of content accesses versus content ranking for Kosarak trace [24] (α = 1.99). Comparison of LRU and moving average with (b) cache size = 100, and (c) cache size = 500.

is predicted by taking the simple average of the most recent m values, i.e. $\hat{Y}_{t+1} = (Y_t + Y_{t-1} + \ldots + Y_{t-m+1})/m$. More complicated prediction models [25], [26], [27] can also be used, however, in this paper we consider SMA for the content demand prediction because it is simple, lightweight and can be easily implemented in CCN routers.

To evaluate the effectiveness of SMA based prediction based caching against the *least recently used (LRU)* based caching, we use a real dataset named *Kosarak* [24] that is widely used and reported in the data mining literature. 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. In our experiments, we consider every single item in serial order. Fig. 2(a) shows the number of times a content has been accessed versus the ordering of the content in the trace. In Fig. 2(a), the contents are sorted (or ordered) based on their frequency in the trace file, where order 1 is the most frequently accessed content. From this figure we can observe that ignoring the far end of the tail, the curve fits a straight line (in the log-log scale) reasonably well, which implies that the content access frequency is proportional to $1/i^{\alpha}$ as stated in equation (1).

Fig. 2(b)-(c) show the comparison of LRU and SMA (ties due to identical $\hat{Y}$ are broken based on the content recency) with cache size 100 and 500 respectively, where the contents are assumed to arrive at one per second. We assume $m = 5$ for Fig. 2. From these figures we can observe that the simple moving average based content access prediction scheme improves the cache hit by $\sim$10-12%. Similar improvements are also observed with other well-known trace files [24], [28]. Such popularity predictions are useful for taking effective caching decisions as discussed in section IV.

Content freshness is another important requirement of any CCN architecture. Contents are updated occasionally in today's Internet, the frequency of which is dependent entirely on the type of the contents. 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. Ensuring content freshness is crucial to serve the clients with up to date information [29]. 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 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 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 fetches the updated contents and broadcasts to it's neighboring routers using a Bloom Filter to purge those contents. Thus further requests for those contents are directed towards the Repo, which sends 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. The details of the message passing between the Repos and the actual servers for maintaining fresh contents is beyond the scope of this paper.

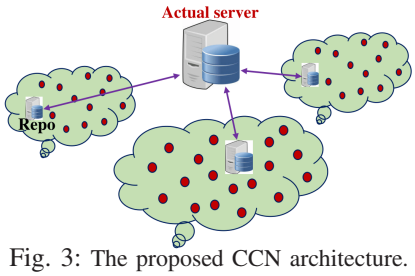


Fig. 3: The proposed CCN architecture.

IV. Neighborhood Aware Caching and Interest Dissemination in 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 [30]. 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,

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.

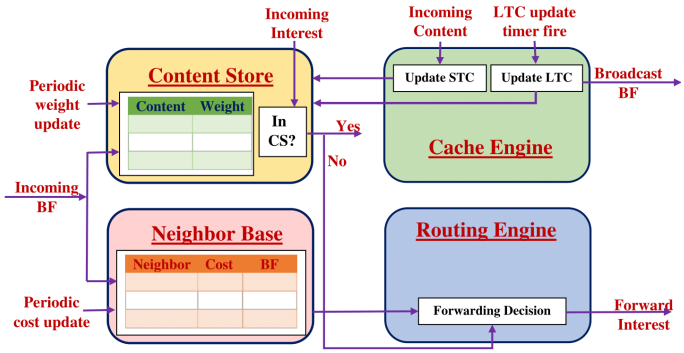


Fig. 4: The overall NACID architecture.

and (b) its predicted content demand and its obsolescence rate. The overall NACID architecture is shown in Fig. 4. 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. 5. 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 number of contents is extremely large or infinite (as new contents are continuously generated), we use Bloom filter (BF) to encode the presence of a content in a router's LTC.

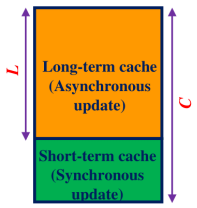


Fig. 5: Two level caching.

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 to check whether a given chunk is within the set.

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. In this manner, the nodes share their *partial* cached chunks (only LTC chunks) information in their neighborhood, with limited broadcast overhead. Given such a mechanism, it is easy to *reactively* cache the incoming chunk if it is not available in the close neighborhood.

Each CCN node runs a caching engine (see Fig. 4) that maximizes the overall *benefit* of its cache, which we define shortly. For STC the caching decision is to check which chunks (if any) are to be replaced, upon arrival of a new chunk. In contrast, the purpose of cache reshuffling is to choose the most useful chunks to store in LTC and broadcast. At any instant t, the benefit (w_i) of a chunk is proportional its *cost-demand factor*, which is the product of the moving average of its access demand $\hat{Y}_t$ and the cost c_t to get it from the nearest neighbor. The chunks with identical cost-demand factor are differentiated based on their recency. Thus $w_i = \alpha c_t \hat{Y}_t + \frac{\beta}{\max(\Delta_i, \varepsilon)}$ $\left(\alpha \gg \frac{\beta}{\varepsilon}\right)$, where Δ_i is the difference between the current time and the time when a chunk was last encountered. The term ε ensures that the second factor cannot be a dominating factor for very small Δ_i. 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 Δ_i). 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:

$$\text{Max} \sum_{i=1}^{\mathcal{M}} w_i.y_i \quad \text{subject to} \sum_{i=1}^{\mathcal{M}} y_i.s_i \leq C, \quad x_i \in \{0,1\} \tag{2}$$

The above problem is identical to the 0-1 Knapsack problem [31] 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:

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.

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 same size, the greedy algorithm converges to the optimal solution.

In our simulations, we assume that all contents-chunks are of equal sizes, which is generally assumed in the literature [26]. 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 [26].

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. Notice that for identical content-chunk sizes, this reactive mechanism just requires a benefit comparison between (a) the newly arrival chunk and (b) the chunk with least benefit in STC, and thus can be done at line speed of the routers.

B. Routing Engine

Another component in Fig. 4 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. For simplicity we assume hop-count as a cost-metric for our simulations. Each router maintains this 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.

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 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. 6 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.

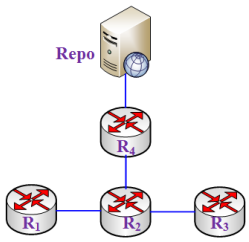


Fig. 6: An illustrative example.

V. Simulation Results

We analyze our proposed NACID scheme using **CCNSIM** [32], which is an application-level simulator for content centric

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.

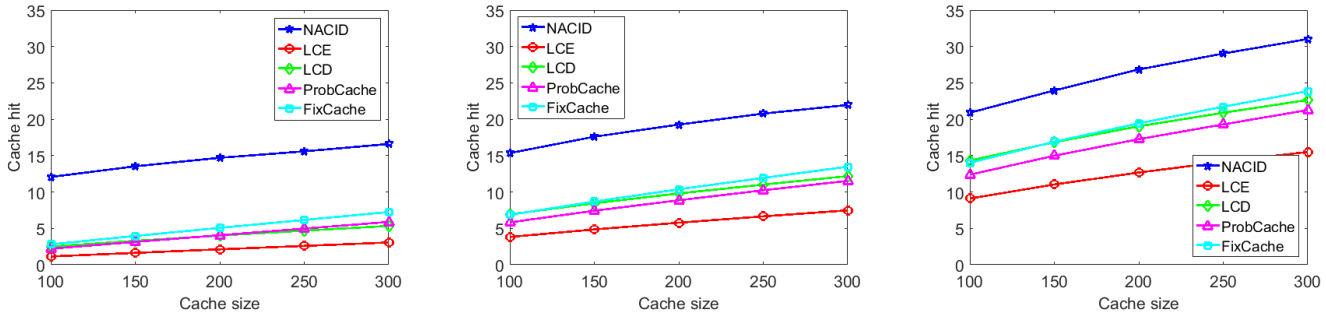


Fig. 7: Comparison of cache hit ratios for different caching schemes, with (a) α = 0.5, (b) α = 0.8, (c) α = 1.

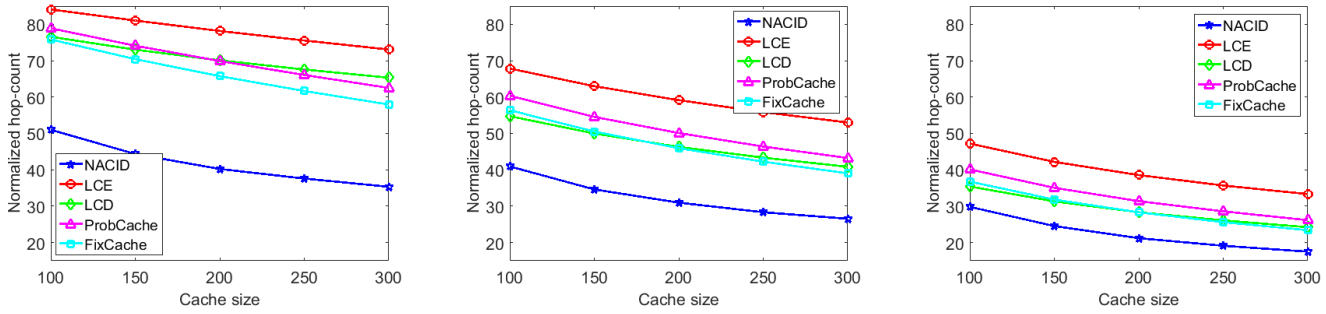


Fig. 8: Comparison of normalized hop-counts for different caching schemes, with (a) α = 0.5, (b) α = 0.8, (c) α = 1.

network based on OMNeT++. We assume a 10×10 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 [33].

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.

We assume that the popularity distribution of the contents is Zipf with decay parameter α. Note that $\alpha = 0$ implies a uniform distribution, and a larger α implies a distribution with shorter tail. The entire cache space is divided among the STC and LTC, with a ratio of 1:3. We assume a total content pool of 5000 with identical content-chunks. As we have assumed identical content-chunks, the cache sizes are defined by the number of chunks that the cache can accommodate. We use α =0.5, 0.8, and 1.0 for the results.

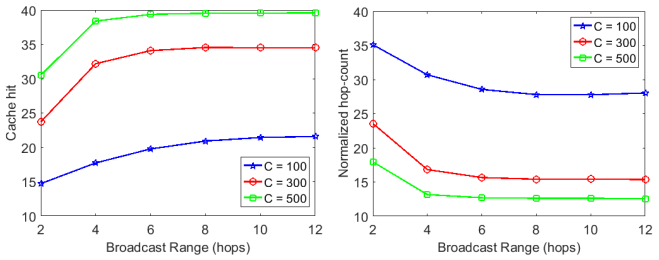
A. Performance comparison with other schemes

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. Higher is the number of bits in the BF lower false positive effects arise due to such collisions. Assume M is the cardinality of the set that needs to be represented using a BF, which is assumed to be the number of contents in the content pool. The false positive probability p is minimized if the length of the BF is optimally chosen to be $m = (-M \ln p)/(\ln 2)^2$ [34]. The corresponding optimal number of hash functions to be used is equal to $k = (m \ln 2)/M$. Using these the values of m and k are chosen to be 5120 bytes and 6 respectively to keep p approximately 0.01. Our current implementation of Bloom filter is borrowed from [35], which uses CRC32 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 total number of hops to the repository that the Interest must walk before getting the chunk.

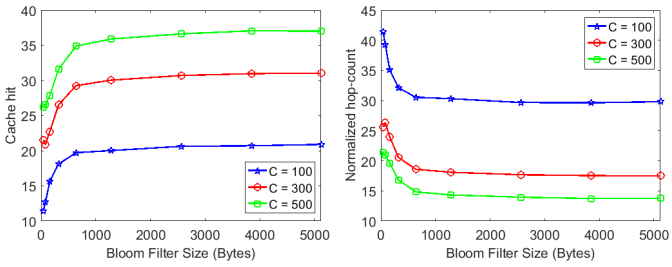
Performance of cache hit percentage: Fig. 7 shows the cache hit percentage of NACID in comparison to other schemes, with the variation of cache sizes. From Fig. 7(a) we can observe that with α = 0.5, NACID improves the cache hit probability up to ∼3 times compared to the other schemes. With higher α (i.e. α = 1), the improvement reduces to ∼5%-10% compared

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.



(a) (b)

Fig. 9: Comparison of (a) cache hit ratios and (b) normalized hop-counts with broadcast range.



(a) (b)

Fig. 10: Comparison of (a) cache hit ratios and (b) normalized hop-counts with different Bloom filter sizes.

to others. This is because for large α, 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 $\sim$5-10%, when the cache size increases from 100 to 300 based on different α. 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 α increases from 0.5 to 1. This is because with the increase in α, more popular chunks are fetched more often, which overall improves the cache hit probability.

Performance of normalized hop-counts: Fig. 8 shows the comparison of the normalized hop-counts of NACID against other proposals. With α = 0.5, NACID reduces the number of hop traversed by $\sim$30% compared to others. Similar improvements are also evident with higher α. This clearly shows the improvement of NACID due its neighborhood awareness. We can also observe that the NHC goes down by $\sim$12%-15% 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 α from 0.5 to 1, the NHC reduces by $\sim$20%. The reason is because with higher α more popular chunks are fetched more frequently, so the cache hit increases and at the same time the number of hops traversed decreases.

The results show that *the NACID algorithm improves the cache hit ratio up to 3 times over all other algorithms, and it does so while also simultaneously reducing the NHC by up to 30%*. This establishes the superiority of our algorithm over previous CCN caching algorithms, with only a small increase 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.

B. Performance of NACID with different tuning parameters

Comparison with different broadcast range: Fig. 9 shows the variation of cache hit ratio and normalized hop-traversal with different broadcast ranges, when α is assumed to be 1. From Fig. 9 we can observe that the cache hit ratio improves by $\sim$5-10%, and the NHC reduces by $\sim$5-7% 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.

Comparison with different BF size: The Bloom filter size plays a significant role in NACID performance due to its false positive effects. Fig. 10 shows the effects of Bloom filter size on the cache hit ratio and NHC, where the α is assumed to be 1. From Fig. 10 we can observe that the hit ratio increases by $\sim$7-10%, whereas the NHC reduces by $\sim$5-10% 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 some real datasets which follows power law distribution. These datasets are publicly available and are widely used in data mining literature. We use four datasets that have diverse characteristics, which is reported in Fig. 11. Other than the Kosarak dataset, we used three other datasets which are described as follows:

Retail: This dataset consists of retail market based data, which are obtained from an anonymous Belgian store. We consider every single item in serial order.

Q148: This dataset if derived from KDD Cup 2000 data, which is the 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, which is the compliments of NASA and the Voyager 2 Triaxial Fluxgate Magnetometer principal investigator, Dr. Norman F. Ness.

We have 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. 12 shows the performance of NACID compared to the other schemes. For Kosarak and Retail datasets, NACID improves

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.

	Count	Distinct items	Min	Max	α
Kosarak	8019015	41270	1	41270	1.9979
Retail	908576	16470	0	16469	1.5533
Q148	234954	11824	0	149464496	1.1104
Nasa	284170	2116	0	28474	2.0735

Fig. 11: Statistical characteristics of the datasets used.

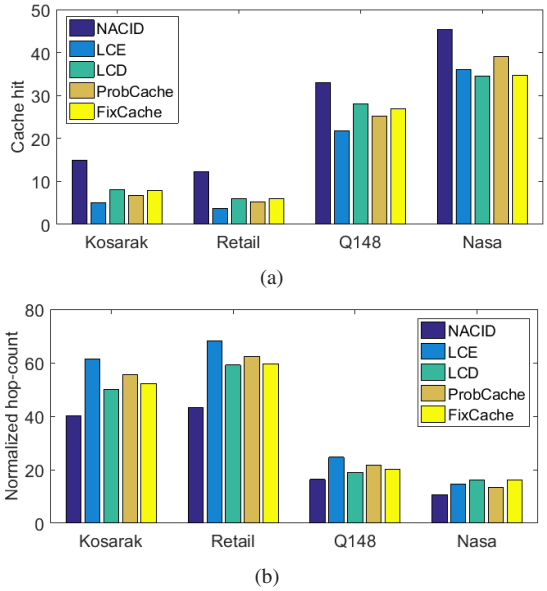


Fig. 12: Comparison of (a) cache hit ratios and (b) normalized hop-counts corresponding to different real datasets.

the cache hit by ∼2-3 times, whereas the hop-count is reduced by ∼10-20%. NACID also shows 5-10% improvement in terms of cache hit and ∼5% improvement in NHC with Q148 dataset, compared to the other schemes. For Nasa dataset, the improvement of cache hit and NHR are ∼10% and ∼5% respectively. The hit ratio of Q148 and Nasa are much more compared to the other two datasets, because of less number of distinct contents in these datasets. For similar reason, the NHR is also less for Q148 and Nasa.

VI. 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 network file system. 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. On the other hand caching in content distribution networks (CDN) are explored in [36], [37]. However 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.

Caching Decision Policy in CCN: Caching in CCN is also well researched topic, however, in most of the proposed scheme a content router does not need to know the cache information in its neighborhood, thus the caching decisions are taken autonomously by the routers. Leave copy down (LCD) [33], Move copy down (MCD) [33], copy with some probability [33] and Probabilistic cache [14] falls in this category. In [12], the authors have argued that the chunks of a file are correlated or fetched in a sequential manner. They have proposed a scheme named WAVE, where the routers exponentially increase the number of chucks cached for a file with the increase in the number of requests. In contrast to the literature, in NACID the CCN routers occasionally forward BF 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 it towards the content store.

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 [38] and 2Q [39]. Least Frequently Used (LFU) is an alternative of LRU to capture the access frequency. ARC [40] 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 [20], [21]. The studies concluded that the content popularity in CCN follows heavy-tailed distributions, which can be modeled as a power-law distribution. They have also observed that a significant portion of the contents are just one-timers. The effect of short-term and long-term content popularity is studied in [41], [42]. The value of the scale-factor of the popularity distributions varies in the literature from 0.6 [22] to 2.5 [23].

Bloom Filter: The Bloom filter data structure has been first introduced by Burton Bloom in 1970 [43]. Since then Bloom filter has been used in various domains including Web caching [44], P2P networks [45], packet routing and forwarding [46], RFID tag identification [47], differential file access in DBMS systems [48] etc. There are different variants of Bloom filters that are proposed in the literature, such as counting Bloom filter [49], compressed Bloom filter [50], deletable Bloom filter [51], hierarchical Bloom filter [52] etc. The use of Bloom filter in CCN has been studied in [53], [54], [55], [56].

VII. 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 filter, and (c) using them for developing a neighborhood-aware two-level caching and interest forwarding

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.

scheme. The simulation results show that the NACID, compared to the existing caching algorithms, effectively increases hit ratio, and at the same time reduces the number of hops for fetching the contents. In future, we want to develop an analytical model for our NACID architecture to study its improvement compared to other caching strategies. We will also study the performance of NACID on different network topologies, along with heterogeneous cache sizes with various content popularity prediction models.

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 *et al.*, "Distributed caching algorithms for content distribution networks," in *IEEE INFOCOM*, 2010, pp. 1478–1486.

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

[4] G.Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[5] B.Ahlgren *et al.*, "A survey of information-centric networking," *IEEE Communications Magazine*, July 2012.

[6] M.Bari *et al.*, "A survey of naming and routing in information-centric networks," *IEEE Communications Magazine*, Dec 2012.

[7] J.Choi *et al.*, "A survey on content-oriented networking for efficient content delivery," *IEEE Communications Magazine*, vol. 49, no. 3, pp. 121–127, 2011.

[8] L.Zhang *et al.*, "Named data networking," *Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[9] A.Wolman *et al.*, "On the scale and performance of cooperative web proxy caching," in *ACM SOSP*, 1999, pp. 16–31.

[10] A. I. T.Rowstron *et al.*, "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 *et al.*, "Cooperative caching: Using remote client memory to improve file system performance," in *USENIX (OSDI)*, 1994, pp. 267–280.

[12] K.Cho *et al.*, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *INFOCOM Workshops*, 2012, pp. 316–321.

[13] Z.Ming *et al.*, "Age-based cooperative caching in information-centric networks." in *INFOCOM Workshops*, 2012, pp. 268–273.

[14] I.Psaras *et al.*, "Probabilistic in-network caching for information-centric networks," in *ACM ICN*, 2012, pp. 55–60.

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

[16] K.Kant *et al.*, "Internet of perishable logistics," *IEEE Internet Computing*, vol. 21, no. 1, pp. 22–31, 2017.

[17] A.Pal *et al.*, "Towards building a food transportation framework in an efficient and worker-friendly fresh food physical internet," *in submission*.

[18] A.Pal *et al.*, "Networking in the real world: Unified modeling of information and perishable commodity distribution networks," in *IPIC*, 2016.

[19] A.Pal *et al.*, "$F^2\pi$: A physical internet architecture for fresh food distribution networks," in *IPIC*, 2016.

[20] P.Gill *et al.*, "Youtube traffic characterization: A view from the edge," in *IMC*, 2007, pp. 15–28.

[21] M.Zink *et al.*, "Characteristics of youtube network traffic at a campus network - measurements, models, and implications," *Computer Networks*, vol. 53, no. 4, pp. 501–514, 2009.

[22] K. V.Katsaros *et al.*, "Multicache: An overlay architecture for information-centric networking," *Computer Networks*, vol. 55, no. 4, pp. 936–947, 2011.

[23] L.Muscariello *et al.*, "Bandwidth and storage sharing performance in information centric networking," in *ACM ICN*, 2011, pp. 26–31.

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

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

[26] S.Li *et al.*, "Popularity-driven content caching," in *IEEE INFOCOM*, 2016, pp. 1–9.

[27] H.Nakayama *et al.*, "Caching algorithm for content-oriented networks using prediction of popularity of contents," in *IFIP/IEEE IM*, 2015, pp. 1171–1176.

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

[29] A. K.Pathan *et al.*, "A taxonomy and survey of content delivery networks."

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

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

[32] R.Chiocchetti *et al.*, "ccnsim: An highly scalable CCN simulator," in *IEEE ICC*, 2013, pp. 2309–2314.

[33] N.Laoutaris *et al.*, "The LCD interconnection of LRU caches and its analysis," *Perform. Eval.*, vol. 63, no. 7, pp. 609–634, 2006.

[34] S.Tarkoma *et al.*, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

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

[36] K.Park *et al.*, "Scale and performance in the coblitz large-file distribution service," in *NSDI*, 2006.

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

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

[39] T.Johnson *et al.*, "2q: A low overhead high performance buffer management replacement algorithm," in *VLDB*, 1994, pp. 439–450.

[40] N.Megiddo *et al.*, "Arc: A self-tuning, low overhead replacement cache," in *FAST*, 2003, pp. 115–130.

[41] Y.Borghol *et al.*, "Characterizing and modelling popularity of user-generated videos," *Perform. Eval.*, vol. 68, no. 11, pp. 1037–1055, 2011.

[42] S.Mitra *et al.*, "Characterizing web-based video sharing workloads," *ACM Trans. Web*, vol. 5, no. 2, pp. 8:1–8:27, 2011.

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

[44] L.Fan *et al.*, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.

[45] H.Cai *et al.*, "Applications of bloom filters in peer-to-peer systems: Issues and questions," *IEEE NAS*, vol. 0, pp. 97–103, 2008.

[46] H.Song *et al.*, "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.

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

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

[49] K.-Y.Whang *et al.*, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, 1990.

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

[51] C. E.Rothenberg *et al.*, "The deletable bloom filter: a new member of the bloom family," *IEEE Communications Letters*, vol. 14, no. 6, pp. 557–559, 2010.

[52] K.Shanmugasundaram *et al.*, "Payload attribution via hierarchical bloom filters," in *ACM CCS*, 2004, pp. 31–41.

[53] W.Quan *et al.*, "Scalable name lookup with adaptive prefix bloom filter for named data networking," *IEEE Communications Letters*, vol. 18, no. 1, pp. 102–105, 2014.

[54] ——, "TB2F: tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking," in *IFIP Networking*, 2014, pp. 1–9.

[55] C.Tsilopoulos *et al.*, "Reducing forwarding state in content-centric networks with semi-stateless forwarding," in *IEEE INFOCOM*, 2014, pp. 2067–2075.

[56] Y.Wang *et al.*, "Advertising cached contents in the control plane: Necessity and feasibility," in *IEEE INFOCOM*, 2012, pp. 286–291.

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY KANPUR. Downloaded on December 20,2021 at 10:16:44 UTC from IEEE Xplore. Restrictions apply.