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, 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 decide when and how to handle the additional caching of content and its eviction. Simulation results show that NACID provides an increase in upto $\sim$5 times of cache hits, and decrease in upto $\sim$30% the number of hops required to get the contents than existing CCN caching policies. We also study different heterogeneous cache memory allocation strategies and show that there is no real incentives for using such cache allocation strategies as opposed to identical cache assignment across the content routers.

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

1 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 time in the last five years. The annual traffic volume is anticipated to increase by 29%. Among the Internet traffic, the video traffic itself account for 86% of all the IP traffic in 2016 [1], which will continue to grow due to the present 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.

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 [16] where the caching decision is made based on whether there is any copy of the content exist 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 the a given delay limit can also be considered. Also in a CCN different links have cost (capacities, traffic volumes, delay

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

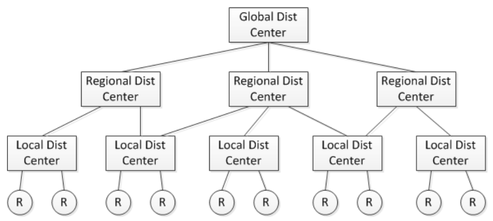


Fig. 1: An abstract model of PCDN. "R" denotes a retailer.

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-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 upto $\sim$5 times, whereas the hop-reduction percentage goes down upto $\sim$30%.

The outline of this paper is as follows. Section 3 proposes the system model, content popularity distribution and network architecture assumed in our scheme. Section 5 introduces the proposed neighborhood aware caching scheme and describes the operation and interaction of the two-level caching mechanism. Section 6 shows the simulation comparison of NACID against other well-known existing schemes. Relevant literatures and discussions are summarized in section 7. Finally, the paper is conclude in section 8.

2 Motivation behind NACID

Interestingly the key motivation behind this work stems from our recent efforts for building an efficient Perishable Commodity Distribution Networks (PCDN) [17], [18], [19], [20]. 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" endpoints, 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 3 and section 5.

3 The System Model

3.1 Content Popularity Distribution

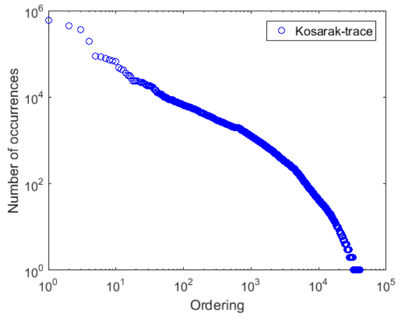
In CCN the content popularity is determined by how often a piece of content is requested. Recent studies [21], [22] show that the users are attracted by only few contents, while others are accessed rarely. Infact a significant portion of the contents are one-timers. Therefore, the content popularity 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 its rank. In a Zipf distribution, out of the population of $\mathcal{M}$ contents, the frequency of the i-th content is given by

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

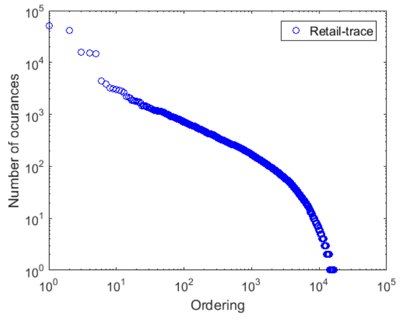
where α is the Zipf exponent and $\mathbf{H}_{\mathcal{M},\alpha} = \sum_{j=1}^{\mathcal{M}} \frac{1}{j^\alpha}$ the generalized harmonic number of order α. Formally in the Zipf distribution, the relative probability of a request for the i-th most popular content is proportional to $\frac{1}{i^\alpha}$. In literature the range of α is varied from 0.6 [23] to 2.5 [24].

By taking logarithmic values on both side in equation(2), we obtain

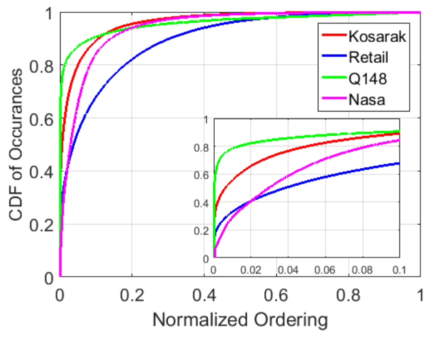
$$\log f(i,\alpha,\mathcal{M}) = \log\left(\frac{1}{\mathbf{H}_{\mathcal{M},\alpha}}\right) - \alpha \log i \quad (2)$$



(a)



(b)



(c)

Fig. 2: Frequency of content accesses versus content ranking for (a) `Kosarak` (α = 1.99) and (b) `Retail` (α = 1.55) traces [25]. (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 Zipf 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. In our experiments, we consider every single item of these traces in serial order. Fig. 4(a)-(b) show the number of times a content has been accessed versus the ordering of the content in the trace. In Fig. 4(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 $1\big/i^{\alpha}$ as stated in equation (2).

3.2 Content Popularity vs Freshness

For Zipf distribution, the cumulative probability that one of the most popular k contents are accessed is given by $\phi(k) \approx \left(\frac{k}{\mathcal{M}}\right)^{1-\alpha}$ [26], when $0 < \alpha < 1$. Thus for larger α, more content requests are concentrated on few popular contents, whereas less popular contents are accessed less frequently. Fig. 4(c) shows the CDF of occurrences of the top $r\%$ contents in four traces (`Kosarak`, `Retail`, `Q148`, `Nasa` obtained from [25]). 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 identifying the hot contents in CCN, which varies both spatially and temporally, are extremely important to take effective caching decisions. We thus model some content popularity prediction schemes in section 4, which are used in content caching in section 5.

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

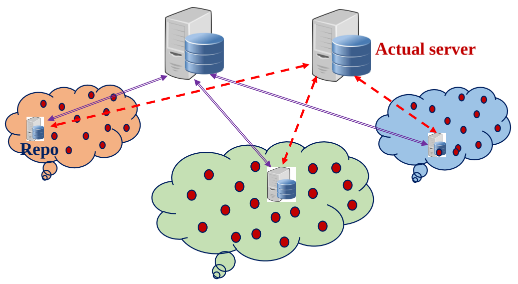
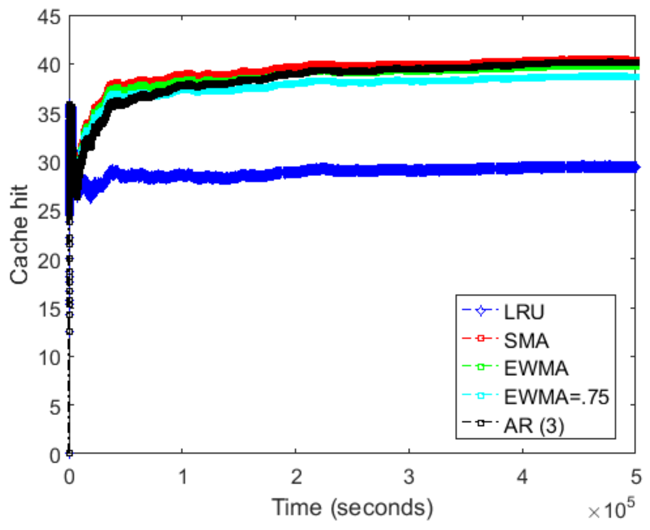


Fig. 3: The proposed CCN architecture.

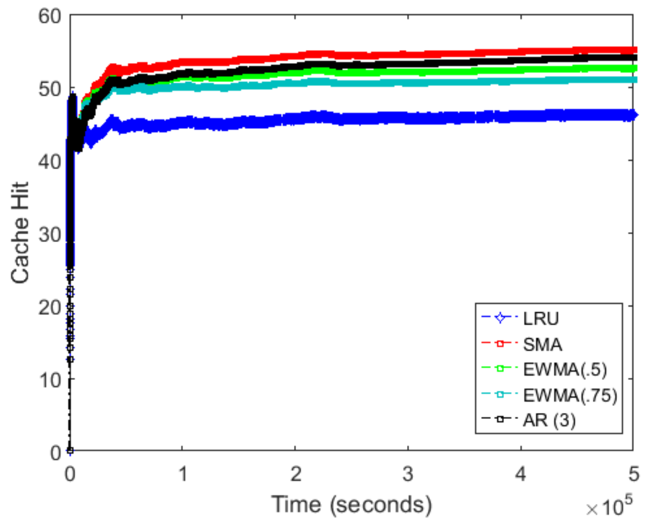
become stale sooner compared to reality shows or movies since they are constantly updated. Also, different types of news have different update rates and useful life, e.g., those concerning a fast moving disaster vs. normal events. Ensuring content freshness is crucial to serve the clients with up to date information [27]. 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 volume 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 broadcasts 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, 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.

4 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



(a)



(b)

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.

regions. Thus the popularities of programs with regional dialects or importance greatly varies spatially and temporally. To predict this dynamic and regional popularity, we first consider few well-known time-series prediction schemes as mentioned below. These prediction models will run at each CCN routers independently to capture the *regional* variation of content popularities.

Simple moving average model (SMA): We first describe a simple content demand prediction model using the *simple moving average model (SMA)* [28]. SMA is used for predicting time series, where the value of Y at time $t+1$ is predicted by taking the simple average of the most recent m values, i.e. $\hat{Y}_{t+1} = (Y_t + Y_{t-1} + \ldots + Y_{t-m+1}) \big/ m$.

Exponentially weighted moving average model (EWMA): EWMA is the most widely used time series prediction model. It weights the past data in an exponentially decreasing manner. Mathematically the value of Y at time $t+1$ is computed by interpolating between the last observed value (Y_t) and the forecast that had been made for it ($\hat{Y}_t$), i.e. $\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t$, where α is the smoothing constant

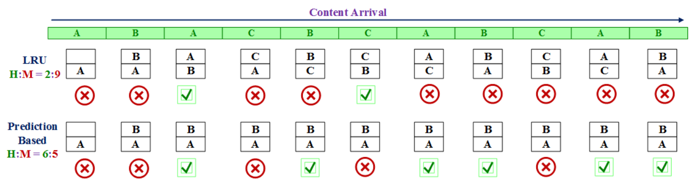


Fig. 5: An illustrative example for comparing LRU and popularity prediction based caching.

in between 0 and 1.

Autoregressive (AR) model: An Autoregressive (AR) model is one of the most popular methods for modeling and predicting future values of a time series [29]. Given a time series Y, an AR model of order p is defined as:

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

where $\alpha_1, \ldots, \alpha_p$ are the parameters of the model and ε is a white noise error term. The error terms, ε_t, are generally assumed to be Gaussian i.i.d. random variables with zero mean and constant variance.

To evaluate the effectiveness of prediction based caching against the *least recently used (LRU)* based caching, we use a real dataset named `Kosarak` [25] that is widely used and reported in the data mining literature. 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, where the contents are assumed to arrive at one per second. 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 improves the cache hit by ∼10-12% in comparison to LRU.

Similar improvements are also observed with other well-known trace files [25], [30]. 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 5.

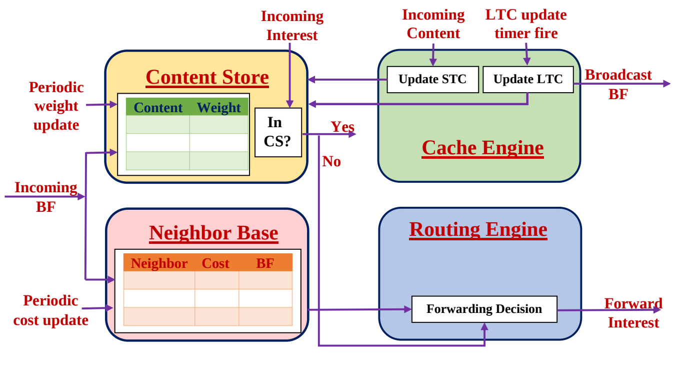


Fig. 6: The overall NACID architecture.

5 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 [31]. 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. 6. The entire scheme is summarized below, by describing the two key modules, named *Cache Engine* and *Routing Engine* that run at each router.

5.1 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. 7. 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.

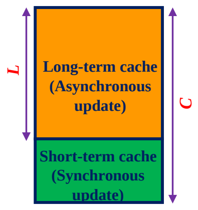

L
Long-term cache (Asynchronous update)
C
Short-term cache (Synchronous update)

Fig. 7: 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. A typical example if Bloom filter is shown in Fig. 8 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.

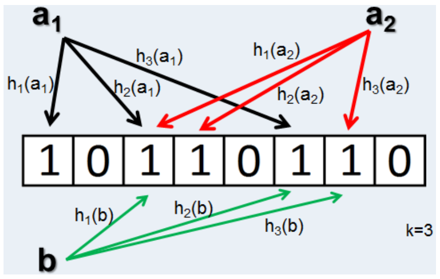


Fig. 8: A typical Bloom Filter.

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. 6) 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:

$$\text{Max} \sum_{i=1}^{\mathcal{M}} w_i.y_i \quad \text{subject to} \sum_{i=1}^{\mathcal{M}} y_i.s_i \leq C, \quad x_i \in \{0,1\} \quad (4)$$

The above problem is identical to the 0-1 Knapsack problem [32] 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 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 [33]. 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 [33].

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 (a) the newly arrival chunk and (b) 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. Next the MIN-HEAPIFY (at the root) is called to maintain the HEAP, which can be done in $\mathcal{O}(\log n)$ time.

5.2 Routing Engine

Another component in Fig. 6 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 costs 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.

5.3 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. 9 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.

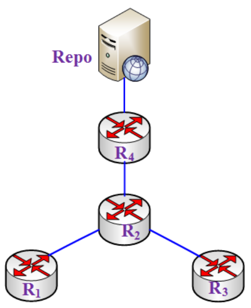


Fig. 9: An illustrative example.

5.4 A Special Case for Infinite Cache

For developing an analytical model we consider the case where the cache has infinite amount of storage. Thus, all the contents that are requested previously by a router is stored in its cache. For simplicity we assume that time is divided into slots, and a content request arrives at a slot. We first want to determine the probability that a content will be found in its local cache at time slot s. At any time slot the i-th chunk is accessed with a probability of $P_i = f(i, \alpha, \mathcal{M})$. Also at s-th time slot, the content i is present in the local cache is given by $\left(1 - (1 - P_i)^{s-1}\right)$. Thus the probability that the request at time slot s will be a hit is given by

$$\mathbb{H}_s^{\text{local}} = \sum_{i=1}^{\mathcal{M}} P_i \left(1 - (1 - P_i)^{s-1}\right) \tag{5}$$

Now let us assume that there are n number of routers (including itself) within the vicinity of one hop from the test router. Thus at the time instance s, the requested content is not found within the one hop neighborhood is equal to the probability that content is not accessed by any of the n routers in the previous $(s-1)$ slots. Thus the probability that a request at time s is found within the one hop neighborhood is

$$\mathbb{H}_s^{1-\text{hop}} = \sum_{i=1}^{\mathcal{M}} P_i \left(1 - (1 - P_i)^{n(s-1)}\right) \tag{6}$$

With this we want to calculate how far a router needs to go to fetch a content considering the *best case* scenario, where the routers have sufficient amount of storage. We assume that the Repo is $(x+1)$ hop away from the test router. With this assumption the expected number of hops after which a test router can find a requested content is given by

$$\begin{aligned} L = {} & \mathbb{H}_s^{1-\text{hop}} \left(1 - \mathbb{H}_s^{\text{local}}\right) + 2\mathbb{H}_s^{2-\text{hop}} \left(1 - \mathbb{H}_s^{1-\text{hop}}\right)\left(1 - \mathbb{H}_s^{\text{local}}\right) \\ & + \ldots + x\mathbb{H}_s^{\text{x}-\text{hop}} \left(1 - \mathbb{H}_s^{\text{local}}\right) \prod_{i=1}^{x-1} \left(1 - \mathbb{H}_s^{\text{i}-\text{hop}}\right) \\ & + (x+1)\left(1 - \mathbb{H}_s^{\text{local}}\right) \prod_{i=1}^{x} \left(1 - \mathbb{H}_s^{\text{i}-\text{hop}}\right) \end{aligned} \tag{7}$$

Fig. 10(a) shows the local hit ratio of a router with the variation of s and α. The total number of contents $\mathcal{M}$ is assumed to be 10^5. With high α, the hit rate quickly reaches to $\sim$1. On the other hand with low α, the router takes significant amount of time (or access requests) to ensure a high hit ratio. This is because with low α, the content popularities are less skewed and thus the improvement of hit ratio is less gradual over time. The hit ratio also increases with the increase in s as the requested contents are available in the router's cache with higher probability. The trend is identical in case of $\mathbb{H}_s^{1-\text{hop}}$. Because of the same reason the expected number of hops a router needs to travel to get a content L is more with less α as shown in Fig. 10(b). With higher s, the contents are available in the router's cache with a higher chance and thus the number of hops traversed L reduces.

Note that in Fig. 10(b) at some s, $\mathbb{H}_s^{1-\text{hop}}$ of α = 0.5 is more than that of α = 1. The reason is that in an infinite cache, a content is found in a cache if it is requested beforehead. If we define a time instance $\mathbb{S}$ as a *saturation point* as the time when all the contents are requested atleast once, then after $\mathbb{S}$ all the contents are found in a router's cache. We can show that $\mathbb{S}_{1.5} > \mathbb{S}_1 > \mathbb{S}_{0.5}$, where the subscripts denote different α values. This is because the access probability of the least popular content (i.e. the $\mathcal{M}$-th content) is less with higher α, i.e. $P_{\mathcal{M},1.5} < P_{\mathcal{M},1} < P_{\mathcal{M},0.5}$. Then the expected number of slots to receive the first access request for the $\mathcal{M}$-th content is given by

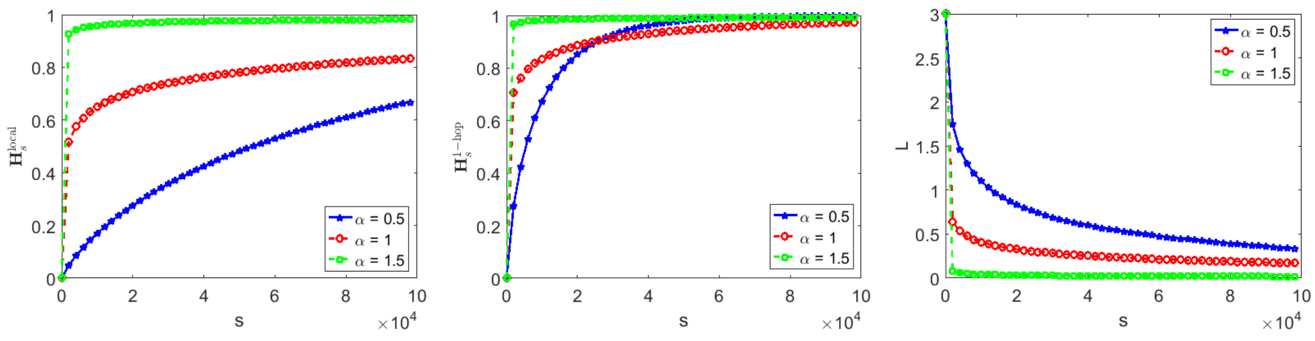
$$\mathbb{S} = P + 2P(1-P) + 3P(1-P)^2 + \ldots = \frac{1}{P} \tag{8}$$

where the subscripts are removed for simplicity. As $P_{\mathcal{M},1.5} < P_{\mathcal{M},1} < P_{\mathcal{M},0.5}$, it follows that $\mathbb{S}_{1.5} > \mathbb{S}_1 > \mathbb{S}_{0.5}$. Thus the saturation point corresponding to α equal to 0.5 is reached before that of 1 and 1.5.

6 Simulation Results

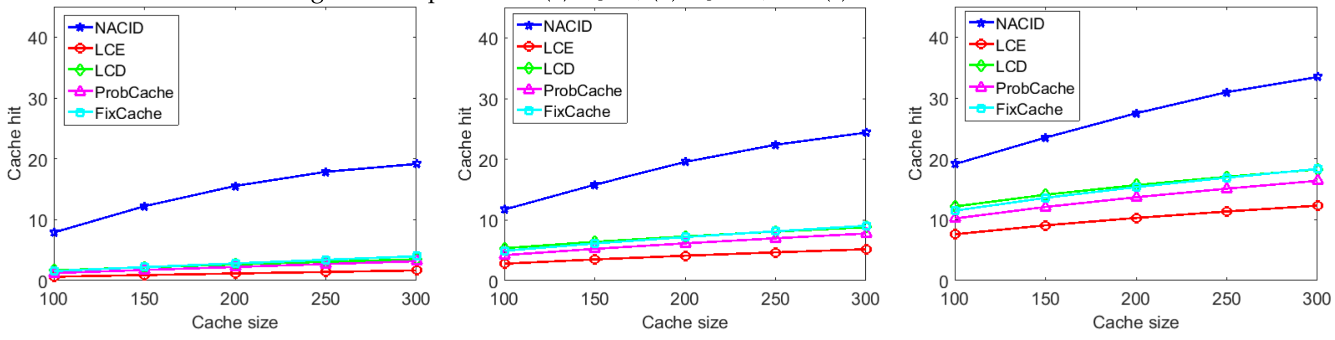
We analyze our proposed CCN scheme using **CCNSIM** [34], which is an application-level simulator for content centric network based on OMNeT++. We assume a 10×10 grid topology consisting of 100 nodes. The content store (Repo), is at a corner of the grid. The content request of a requesting node is assumed 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.



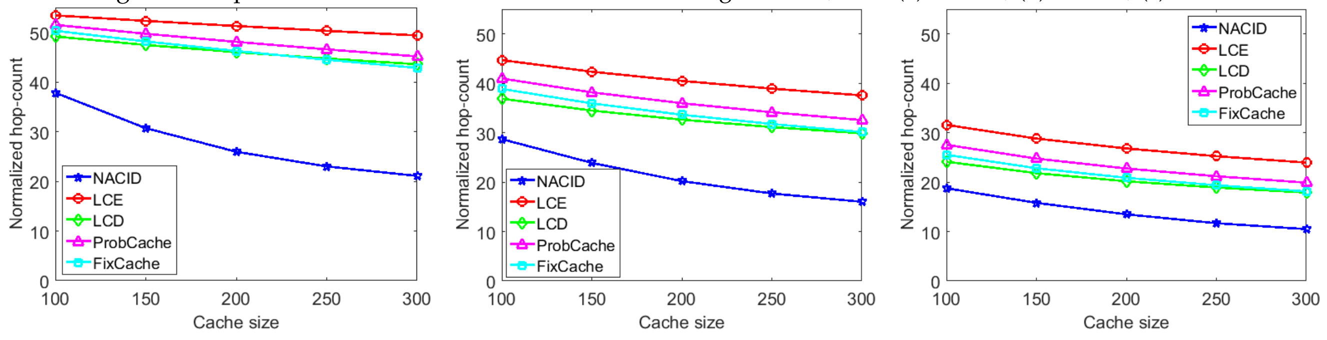
(a) (b) (c)

Fig. 10: Comparison of (a) $\mathbb{H}_s^{\text{local}}$, (b) $\mathbb{H}_s^{1-\text{hop}}$, and (c) L with different s.



(a) (b) (c)

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



(a) (b) (c)

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

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.

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.

6.1 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 $m = (-M \ln p) \big/ (\ln 2)^2$ [36]. The corresponding optimal number of hash functions to be used is equal to $k = (m \ln 2) \big/ M$. 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 [37], 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 rep-

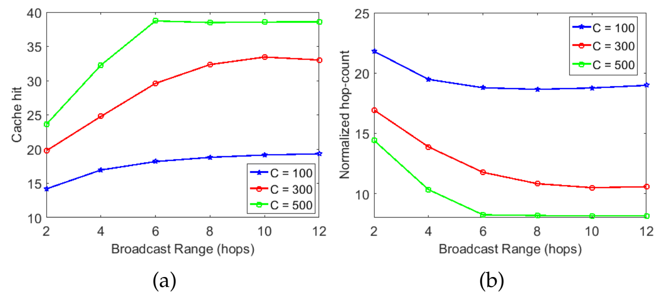


Fig. 13: Comparison of (a) cache hit ratios and (b) normalized hop-counts with broadcast range.

resents 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 α = 0.5, NACID improves the cache hit probability upto $\sim$5 times compared to the other schemes. With higher α (i.e. α = 1), the improvement reduces to $\sim$8%-15% compared 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$10-14%, 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. 12 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$6%-8% 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$10-20%. 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 upto 5 times over all other algorithms, and it does so while also simultaneously reducing the NHC by upto 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.

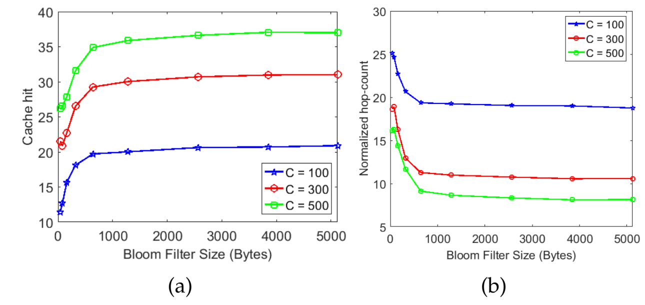


Fig. 14: Comparison of (a) cache hit ratios and (b) normalized hop-counts with different bloom filter sizes.

6.2 Performance of NACID with different tuning parameters

6.2.1 *Comparison with different broadcast range*

Fig. 13 shows the variation of cache hit ratio and normalized hop-traversal with different broadcast ranges, when α is assumed to be 1. From Fig. 13 we can observe that the cache hit ratio improves by $\sim$5-15%, 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.

6.2.2 *Comparison with different BF size*

The bloom filter size plays a significant role in NACID performance due to its false positive effects. Fig. 14 shows the effects of bloom filter size on the cache hit ratio and NHC, where the α is assumed to be 1. From Fig. 14 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.

6.3 Comparison with real datasets:

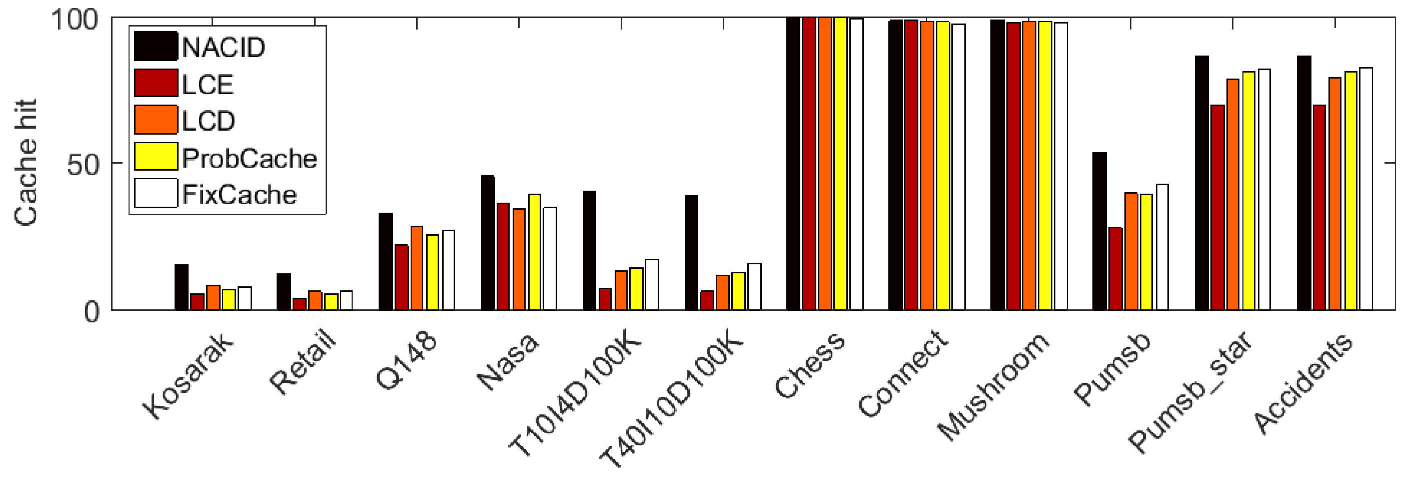
We next compare NACID with others using some real datasets which follows power law distribution. These datasets are publicly available and are widely used in data mining literature. We use twelve datasets that have diverse characteristics, which is reported in Fig. 15. Other than the `Kosarak` and `Retail` datasets, we used several other datasets which are described as follows:

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

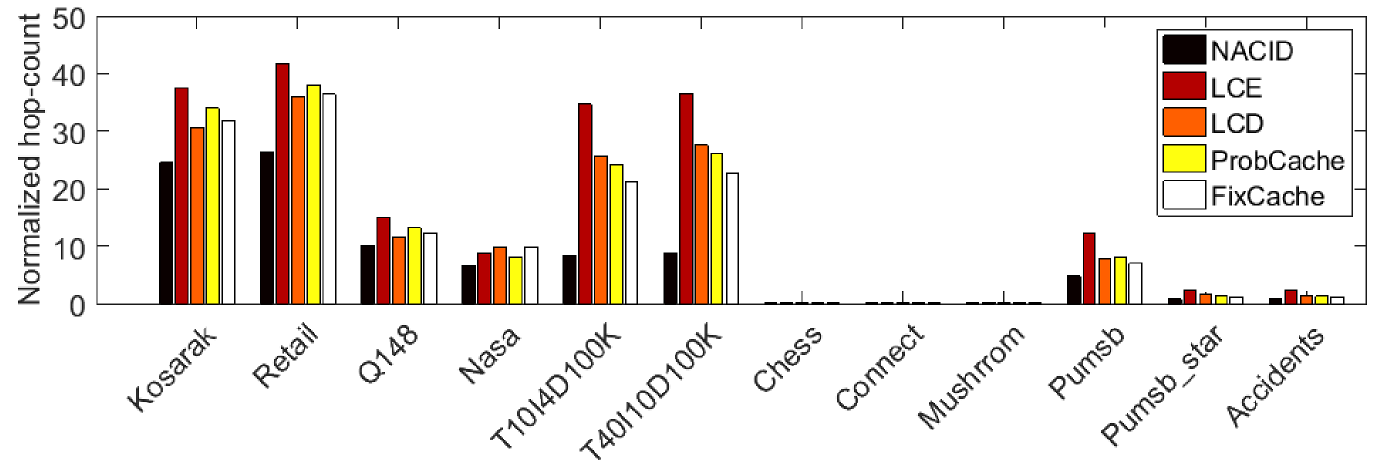
Nasa: This dataset is derived from the "Field Magnitude" and "Field Modulus" attributes from the Voyager 2 spacecraft Hourly Average Interplanetary Magnetic Field Data, which is the compliments of NASA and the Voyager 2 Triaxial Fluxgate Magnetometer principal investigator, Dr. Norman F. Ness.

	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
α	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. 15: Statistical characteristics of the datasets used.



(a)



(b)

Fig. 16: Comparison of (a) cache hit ratios and (b) normalized hop-counts corresponding to different real datasets.

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 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 [25], [38]. `Pumsb` and `Pumsb_star` datasets contain population and housing related census data.

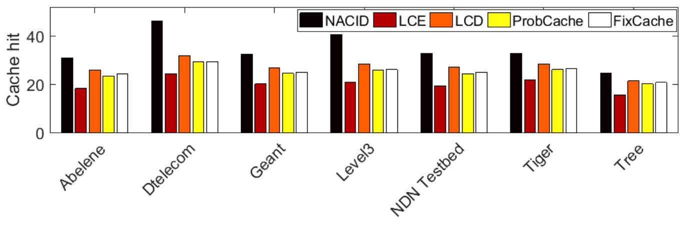
Accidents: This dataset is donated by Karolien Geurts and contains anonymized traffic accident data [25], [39].

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. 16 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 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 performances are similar for `Chess`, `Connect` and `Mishroom` datasets. For the other datasets (i.e. `Pumsb`, `Pumsb_star` and `Accidents`), NACID improves the cache hit and NHC by ~6-26% and ~1.5 times respectively.

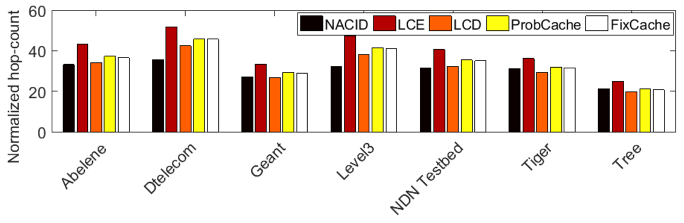
The hit ratio of `Chess`, `Connect` and `Mushroom` are much more compared to the other two datasets, because of less number of distinct contents in these datasets. For similar reason, the NHR is also less for these datasets. Among the others, `Pumsb_star` and `Accidents` perform significantly better, mainly because of lesser number of distinct contents and/or higher α.

	$\|V\|$	$\|E\|/\|V\|$	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. 17: Statistical characteristics of the network topologies.



(a)



(b)

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

6.4 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. 17 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. The purpose of this study is to depict the performance of NACID as opposed to others in various CCN architectures.

From Fig. 18 we can observe that NACID improves the hit ratios by ∼5-15% whereas decreases the NHC by upto ∼15% 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 a 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 even if not in its local cache.

6.5 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 distributed the cache memory by analyzing the level of centrality of the routers within a network. As the central routers of a network serves more content requests, they are assigned more cache spaces 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 $c(x) = \frac{1}{\sum_y d(y,x)}$, 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 σ_{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 serves more requests, and thus putting more cache sizes to them will be beneficial. We thus devise a metric, named Repo-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.

We assume that the total cache size of the topology is fixed and is assumed to be C_{tot}. In case of homogeneous caching (or identical caching **I**), we divide the cache sizes 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

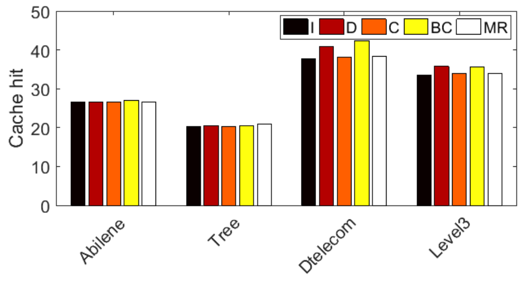
$$C_i = \left\lceil \frac{X_i}{\sum_{j\in V} X_j} C_{\text{tot}} \right\rceil \qquad (9)$$

where X_i is the value of router i depending on which caching strategy (**D**, **C**, **BC**, **MR**) is adopted.

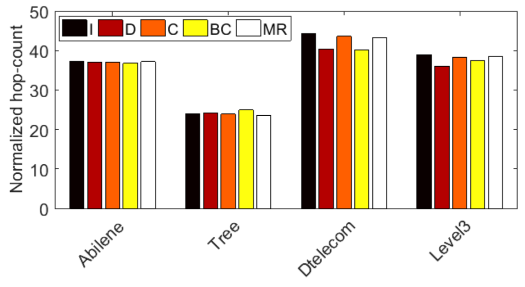
Fig. 19 shows the performance of NACID with heterogeneous caching schemes. We assume α = 1 for this set of figures, and C_{tot} is assumed to be $100\times|V|$. From this figure we can observe that the performance of NACID does not change significantly with heterogeneous cache distribution. Only a modest performance gain of <2% (in cache hit) is observed in case of Dtelecom and Level3 compared to its homogeneous counterpart. Similar findings are also observed in [40]. 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.

6.6 Analytical Model for Homogeneous and Heterogeneous Caching

To validate the performance of homogeneous and heterogeneous caching in general, we develop an analytical model on a linear topology consisting of $N-1$ routers. Assume that R_1 is N hops away from the Repo. Clients are attached



(a)



(b)

Fig. 19: Comparison of (a) cache hit ratios and (b) normalized hop-counts in case of heterogeneous caching.

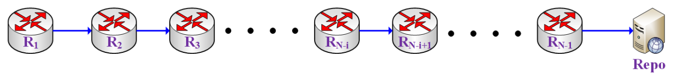


Fig. 20: Illustration of a linear topology for modeling $\hat{L}$.

directly to the individual routers. Also for simplicity let us assume that the contents are independently cached in the routers. Below we describe the analytical model of there types of cache assignment strategies.

Homogeneous Caching: First let us assume that the router-caches are of identical sizes. The probability of a hit and miss are denoted by H and $\overline{H}$ respectively. Here $H = \frac{\mathbf{H}_{C,\alpha}}{\mathbf{H}_{\mathcal{M},\alpha}}$, and $C = \frac{C_{\text{tot}}}{N-1}$. Thus the number of hops an Interest packet from a client of R_{N-i} needs to traverse is given by

$$\begin{aligned} L_i^{HO} &= H + 2\overline{H}H + 3\overline{H}^2 H + \ldots + i\overline{H}^{i-1}H + (i+1)\,\overline{H}^i \\ &= \sum_{l=1}^{i} l\overline{H}^{l-1}H + (i+1)\,\overline{H}^i = \frac{1-\overline{H}^{i+1}}{H} \end{aligned} \tag{10}$$

Thus the average number of hops traversed in case of homogeneous caching is given by

$$\hat{L}^{HO} = \frac{\sum_{i=1}^{N-1} L_i^{HO}}{N-1} = \frac{(N-1)\,H - \left(1-\overline{H}^{N-1}\right)\overline{H}^2}{(N-1)H^2} \tag{11}$$

Heterogeneous Caching: We next consider the scenario where the routers that are closer to the Repo have higher caches. Assume that R_{N-i} which is i-hops away from the Repo has a cache size of $C_{(N-i)} = (N-i)\,\beta C_{\text{tot}}$, where $\sum_{i=1}^{N-1}(N-i)\,\beta = 1$, and its corresponding hit and miss probabilities are given by $H_{(N-i)}$ and $\overline{H}_{(N-i)}$ respectively. Thus $H_{(N-i)}$ is given by $\frac{\mathbf{H}_{C_{(N-i)},\alpha}}{\mathbf{H}_{\mathcal{M},\alpha}}$. In this case the number

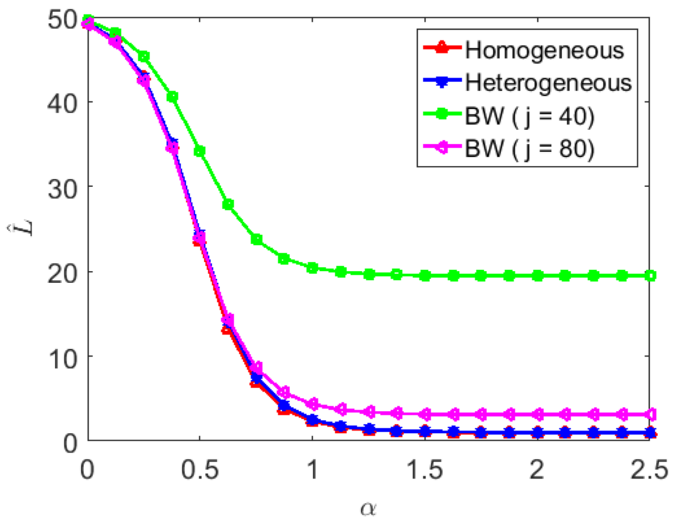


Fig. 21: Comparison of different cache size assignment.

of hops traversed by an Interest packet of R_{N-i}'s client is given by

$$\begin{aligned} L_i^{HE} &= H_{(N-i)} + 2\overline{H}_{(N-i)}H_{(N-i+1)} + 3\overline{H}_{(N-i)}\overline{H}_{(N-i+1)}H_{(N-i+2)} \\ &+ \ldots + (i+1)\overline{H}_{(N-i)}\overline{H}_{(N-i+1)}\ldots\overline{H}_{(N-1)} \end{aligned} \tag{12}$$

Black-White Caching: In Black-White (BW) caching, we assume that the routers from R_{N-j} to R_{N-1} has identical cache sizes of $\frac{C_{\text{tot}}}{j}$, whereas others have no caches. In this scenario, the number of hops an Interest packet of R_{N-i}'s client needs to travel is given by

$$\begin{aligned} L_i^{BW} &= L_i^{HO} \quad \text{if } \; i \le j \\ &= (i-j) + L_j^{HO} \quad \text{if } \; i > j \end{aligned} \tag{13}$$

Thus the average number of hops for BW caching is given by

$$\hat{L}^{BW} = \frac{\frac{(N-j)(N-j+1)}{2} - 1 + (N-j-1)\left(\frac{1-\overline{H}^{j+1}}{H}\right) + \sum_{i=1}^{j}\frac{1-\overline{H}^{i+1}}{H}}{N-1} \tag{14}$$

Fig. 21 shows the comparison of $\hat{L}$ for different cache size assignment. For this figure we assume that N =100, $\mathcal{M}$ = 10^5 and C_{tot} = 10^4. From this figure we can observe that the performance of homogeneous and heterogeneous cache assignment is almost identical. With BW caching, average number of hops traversed is higher especially for higher α. This is because the Interests generated from the clients attached to the routers with no caches, need to traverse several hops before reaching some routers with caches. As expected, with the increase of j the BW cache assignment strategy coincides with the homogeneous caching strategy.

7 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 [41], [42]. 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) [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 chucks cached for a file with the increase in the number of requests. In contrary to these 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 [43] and 2Q [44]. Least Frequently Used (LFU) is an alternative of LRU to capture the access frequency. ARC [45] 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 Polularity in CCN: Content popularity distribution is studied extensively in [21], [22]. 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 [46], [47]. The value of the scale-factor of the popularity distributions varies in the literature from 0.6 [23] to 2.5 [24].

Bloom Filter: The bloom filter data structure has been first introduced by Burton H. Bloom in 1970 [48]. Since then bloom filter has been used in various domains including Web caching [49], P2P networks [50], packet routing and forwarding [51], RFID tag identification [52], differential file access in DBMS systems [53] etc. There are different variants of bloom filters that are proposed in the literature, such as counting bloom filter [54], compressed bloom filter [55], deletable bloom filter [56], hierarchical bloom filter [57] etc. The use of bloom filter in CCN has been studied in [58], [59], [60], [61].

8 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 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 contents. This performance improvement is consistent across different network topologies, as well as for different frequently 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 proves that heterogeneous caching strategies can only affect the performance gain marginally and thus is 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 *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] A.Pal *et al.*, "NACID: A neighborhood aware caching and interest dissemination in content centric networks," in *ICCCN*, 2017, pp. 1–9.

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

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

[19] ——, "Networking in the real world: Unified modeling of information and perishable commodity distribution networks," in *IPIC*, 2016.

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

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

[22] 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.

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

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

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

[26] L.Breslau *et al.*, "Web caching and zipf-like distributions: Evidence and implications," in *IEEE INFOCOM*, 1999, pp. 126–134.

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

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

[29] R.Aufrichtig *et al.*, "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.

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

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

[32] 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.

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

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

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

[36] 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.

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

[38] D.Burdick *et al.*, "Mafia: A maximal frequent itemset algorithm," *IEEE Transactions on Knowledge & Data Engineering*, vol. 17, no. 11, pp. 1490–1504, 2005.

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

[40] D.Rossi *et al.*, "On sizing CCN content stores by exploiting topological information," in *IEEE INFOCOM*, 2012, pp. 280–285.

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

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

[43] 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.

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

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

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

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

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

[49] 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.

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

[51] 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.

[52] 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.

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

[54] 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.

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

[56] 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.

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

[58] 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.

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

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

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