A NEW EFFICIENT CACHE REPLACEMENT STRATEGY FOR NAMED DATA NETWORKING

Saad Al-Ahmadi

Department of Computer Science, King Saud University, Riyadh, Saudi Arabia

ABSTRACT

The Information-Centric Network (ICN) is a future internet architecture with efficient content retrieval and distribution. Named Data Networking (NDN) is one of the proposed architectures for ICN. NDN's in-network caching improves data availability, reduce retrieval delays, network load, alleviate producer load, and limit data traffic. Despite the existence of several caching decision algorithms, the fetching and distribution of contents with minimum resource utilization remains a great challenge. In this paper, we introduce a new cache replacement strategy called Enhanced Time and Frequency Cache Replacement strategy (ETFCR) where both cache hit frequency and cache retrieval time are used to select evicted data chunks. ETFCR adds time cycles between the last two requests to adjust data chunk's popularity and cache hits. We conducted extensive simulations using the ccnSim simulator to evaluate the performance of ETFCR and compare it to that of some well-known cache replacement strategies. Simulations results show that ETFCR outperforms the other cache replacement strategies in terms of cache hit ratio, and lower content retrieval delay.

KEYWORDS

Information-Centric Networking (ICN), Named-Data Networking (NDN), In-network caching, cache replacement.

1. INTRODUCTION

The internet was designed to be an end-to-end connection, with the simple purpose of connecting two computers to transmit data. This model is a host (or producer) centric communication model based on the location of the hosting node (IP address). The consumer must know the IP address of the producer directly or indirectly (using DNS system) to obtain the required content location [1]. This model limits the growing demand for the internet to satisfy the tremendous number of content requests. Also, the need for efficient distribution network increases as the number of users and connected devices increases. A user or a device can be a content producer or content consumer. Information-Centric Networking (ICN) is a content-based internet architecture that solves the distribution problems in an IP-based network [2][3]. ICN represents a shift from host-centric communication to the named-content system and focuses on the data itself rather than its location [4]. There are several proposed architectures for ICN such as Content-Centric Networking (CCN), Named Data Networking (NDN), Publish-Subscribe Internet Technology (PURSUIT), Data-Oriented Network Architecture (DONA), COntent-centric inter-NETwork (CONET), Network of Information (NetInf)/Scalable and Adaptive Internet Solutions (SAIL), CONVERGENCE, and MobilityFirst [5]. In ICN, the content name should be location-independent, globally unique, and persistent. Content name is expressed as a flat string name, hierarchical string name, or any attribute-value based naming. Hence, existing Internet routing protocols have to be replaced by new protocols that route packets using the content name rather than its location. In-network caching, diversification, replication, and freshness are among issues to be considered in ICN communication for efficient content retrieval.

The Content-Centric Networking (CCN) architecture refines and maintains ICN structures for future internet architecture. Named Data Networking (NDN) [6] is an active project that implements CCN architecture and is supported by the Future Internet Architecture program. NDN architecture allows functionality that benefits the user, such as in-network caching and multipath forwarding. In this paper, we focus on NDN as one of the funded ICN projects to realize future internet architecture. NDN divides data into chunks. The consumer node initiates communication and sends an interest packet looking for specific content by its name. Intermediate nodes can reply immediately to interest packets if the content is available in caches; otherwise, the interest will be forwarded until it reaches the producer node where it replies with the requested content.

Using caching in NDN network improves network performance by increasing content availability, avoid producer bottlenecks, reduce upstream traffic, and reduce downstream latency [7], [8]. Each node in the network can cache data chunks in its cache store (content store) based on certain caching decision strategy. If the cache is full, then the node will decide to replace one of the cached data with new arrived content based on certain replacement policy. There are several caching decisions like Leave Copy Everywhere (LCE), Leave Copy Down (LCD), and Prob Cache [9]. Also, there are several replacement strategies like Least Recently Used (LRU), Least Frequently Used (LFU), First In First Out (FIFO), and Random [10] [11].

The proposed strategy in this paper is an enhancement of the LFU replacement strategy with a time cycle factor, called Enhanced Time and Frequency Cache Replacement strategy (ETFCR). It calculates the weighted popularity factor for each chunk and sorts them accordingly. The popularity is defined by a time cycle between the last two requests of each chunk that inversely proportional to the number of hits. Each time a hit occurs, the new weighted popularity added to the previous one. ETFCR evaluated using ccnSim and GEANT network topology [12]. We measured ETFCR performance and compared it versus LRU and LFU strategies. ETFCR comparison results show improvement in number of hits and number of hubs between the consumer node and the content-full filling node. These improvements lead to reduced network fetch time and delay.

The rest of the paper is organized as follows: Section 2 introduces the necessary background of NDN with emphasis on in-network caching and replacement strategies. Section 3 presents the current research literature in cache replacement strategies with a comparison table among some selected papers. Section 4 presents the details of the proposed method, ETFCR, and how it combines time and frequency factors to create efficient cache replacement algorithm. A detailed comparison with some well-known cache replacement strategies highlights the efficiency of the ETFCR is shown in section 5. Section 6 concludes the paper.

2. Background

There are two types of packets (messages) in the NDN: interest and data packets. The interest packet is a request for specific data identified by its name or prefix. The Data packet contains the name and the content of the data along with the content signature. Nodes in NDN network can play at any time one of these roles:

- Consumer (subscriber): is a node that sends an interest packet to request specific content.
- Producer publisher): is a node that produces and stores the requested content by consumers in its cache.
- Router: is a node that routes the interest packet and forwards data packets.

Also, each node in NDN has three data structures [12]:

- Pending Interest Table (PIT): is a table that stores the forwarded interests that did not receive the requested data.
- Forward Information Base (FIB): is a table that works as a routing table mapping interest packets to the selected forwarding output interface.
- Content Store (CS): is a table that locally stores or caches the received data chunks.

The consumer node sends an interest packet to its neighbors, requesting specific data by name. Every neighbor node either reply by the content from its local cache or forward the interest. The process continues until the requested data chunk is found in one of the middle nodes' repositories or until it reaches the producer. The content hosting node will send the requested chunk back in a reverse path to the consumer. When a node receives an interest packet from one of its interfaces, it checks the CS for matching data. If there is a match, it sends the interest back to the requested node through the same interface where the interest received. If the data chunk is not existing in CS, then it checks if there is a packet inside the PIT to match the name of interest. The interest is forwarded and wait for the requested content. In case there is no matching entry in both CS and PIT, the node forwards the interest using FIB and creates a new entry in PIT. The interest is forwarded hop-by-hop until it reaches a node with a cached data chunk or reaches a repository as in [13]. The required chunk sent back following the same path of interest packet by checking the downstream interfaces of PIT entries with the same name. Also, it will delete those PIT entries [2].

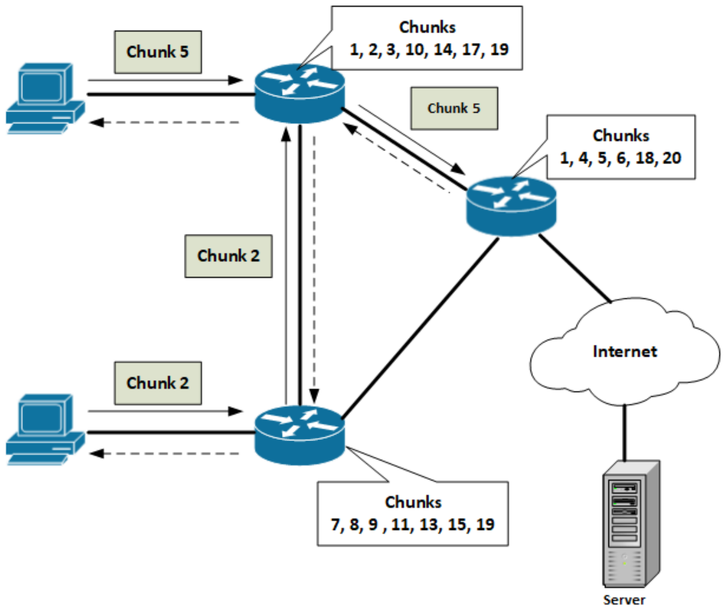


Figure 1. Forwarding interest and receiving cached data chunk

NDN uses hieratically structured and application-dependent names. Each piece of content has a unique name that the node used for forwarding and routing the packets [14]. The uniqueness of names guarantees matching the requested data when this name exists in the producer node. For example, the name of a CNN news homepage content for June 10, 2014, might be: /ndn/cnn/news/2014june10/index.htm. When the content is too large, it is divided into multiple chunks with different names [15]. Caching data chunks in CS, it is playing an important role to improve the performance in NDN. Both IP routers and NDN routers cache the data. In IP routers, the cached data cannot be reused while NDN routers utilize cached data to fulfill requests. IP routers cache data for queueing and scheduling only [14].

One of the main features in NDN is managing nodes caches by caching decision strategy and caching replacement strategy [16]. These two strategies helping in storage utilization of storage and delivering the content efficiently:

a) Caching Decision Strategies:

Caching is about deciding to store a chunk in the node's storage when it arrives or just forward the chunk without storing it. Caching decision strategies can be classified according to their path location as off-path caching or on-path caching [17].

- off-path caching: is about storing the data chunks in the node's storage to increase the chunk's availability regardless of the path of its interest. Data chunks can be in a node's storage even if it is not on the traveled path. The decision made according to multiple information such as cache availability, network traffic, or content popularity [8].
- on-path caching: caching is performed on the same path as the request. data packets are cached along the path to the consumer [1], [8]. when a router receives an interest packet, the router checks to see if the requested data packet is saved in the CS. The consumer receives the data packet directly if the data packet is on the same router. Therefore, the interest packet never reaches the producer, and the consumer has already retrieved the desired data packet.

Different caching decision strategies categorized into multiple ways; one of them presented in [8]:

- Probabilistic Caching: this technique is based on the probability of creating a replica of the data chunk. Examples: Leave-Copy Everywhere (LCE), Random Probabilistic Caching, and ProbCache.
- Graph-based Caching: this technique is based on the location of the node and the topology of the network. Examples: Edge Caching, Leave-Copy Down, and Betweenness Centrality.
- Popularity-based Caching: this technique is based on the calculated popularity of the chunk. Examples: Standalone-popularity Caching and Static-popularity Caching. Standalone-popularity uses a counter for the popularity ranking while Static-popularity uses threshold ranking.

b) Replacement Strategies:

In-network caching includes a cache replacement policy when an entire CS of a router becomes full. Therefore, replacement policy plays a significant role in the caching of ICN architecture. Such policies provide empty storage for incoming cached content by removing previous content [18]. When the cached packet deleted without satisfying any interest at their lifetime in the CS, this denotes inefficient cache. That is why in-network caching objective to improve data dissemination in the network in efficient manner. Router caches inside the NDN network have limited capacity. This limitation is the main challenge of caching. Replacement strategies are about deciding which stored chunk should be removed from the node cache so that the new chunk stored in place.

With LRU algorithm, the replacement of content in the CS based on how recently the content was used. LRU allows cache replacement, when necessary (CS is full), to remove the data content that has not used for the most extended duration of time. LFU is a replacement algorithm tries to cache only the most popular data content. When an eviction is necessary, this policy deletes the content with the smallest access value. Therefore, the LFU gains better cache hit compared to LRU. FIFO deletes the oldest content from a CS when needed. With the Most Frequently Used (MFU), replacement policy, a frequency of appearance distribution across the frequencies is created based on the records of the number of times each frequency has been used. The value of the highest priority assigned to the most frequently used. Particular nodes, including complex data structures, were the reasons for developing the randomized policies. Producing a simple random replacement (RR) policy directly replaces one of the stored. These replacement strategies are based on a single factor (time for example) and they are simple and easy to implement yet they are inefficient and have large number of cache misses. New cache replacement strategies proposed in the literature take extra network information in their cache eviction decision. Extra network information could be the distance from the source, user priority, data priority, content distribution, and network traffic. Packet arrival time is an important caching parameter yet it is considered as an insufficient parameter as high eviction cache rate raises. Many cache replacement strategies use the popularity or the priority of the content chunk [19].

- Content popularity: each data chunk has a weighted popularity value along with data content and name. The weighted popularity value is calculated using number of cache hits. Other factors may be used in the weighted popularity formula for fine-tuning.
- Content priority: this attribute gives more value to significant data chunks and aims to lower their retrieval time. A content priority value is assigned to each data chunk. Data chunks with high priority will have a high probability of being cached and become highly available.

3. LITERATURE REVIEW

In-network caching affects profoundly network performance as it saves time and increases throughput. The available space for caching is limited, thus caching decisions should use it in the best way. Choosing the right replacement strategy can affect the hit ratio and reduce the transmission delay.

The most straightforward replacement strategy in NDN among general replacement strategies is FIFO. In FIFO, the first stored chunk is replaced by the last arrived chunk. Another simple replacement strategy is the Random Replacement strategy (RR). In RR, a randomly chosen chunk is replaced by the newly arrived chunk. However, RR is simple but works efficiently with complex caching decisions. The most widely used replacement strategies are LRU and LFU [20]. In LRU, the hit ratio of the recently used chunks increased, and the old used chunks are evicted to make space for the new chunks. One of LRU advantages is a short execution time [20]. FIFO, RR, and LRU do not take into account the network parameters such as the distance (number of hops) between the producer and the consumer. Also, data priority is not part of their decision algorithm [4]. LRU does not consider the popularity of the data as LFU. Thus if there is an old data chunk that was popular and was not requested anymore, it will remain stored until another data chunk becomes more popular with time [21].

The literature is rich with many replacement strategies with enhanced performance over those general policies. The proposed strategy in [4] called Universal Caching strategy (UC) aims to increase cache hit probability and reduce the total delay by proposing Content Metric System (CMS). CMS uses the function CM(k) (1) calculated for each chunk arriving at any node according to different parameters:

$$CM(k) = f(D(k), F(k), R, P(k)) \quad (1)$$

where, D(k) = Distance from the original source of the content k, F (k) = Frequency of Access of content k, R = Reachability of an ICN router n, P(k) = User given priority of content k;

Every time a new request arrived F(k) value will be increased by one. R indicates the node position where the crowded and centrally nodes have higher values of R, and the edge located nodes have low values. Each time a new request for a chunk came, the CM value of the chunk will be updated. If the node cache is full, then the chunk with the lowest CM value will be replaced by a new arriving chunk. If there is a large requested chunk with high CM value, then the new data chunk cannot be replaced. For that problem, they proposed a solution for the chunk that has not accessed for a specific amount of time called cache refresh time (T) then it will be replaced. After the evaluation, the result showed that the hit ratio increased comparing to FIFO and LRU.

The Least Value First (LVF) strategy used in [22] depends on three parameters for the function (2) that calculates the value of each arrived chunk as:

$$Value_{NDO_i} = \alpha . D_i^{'} + \beta .\ Pop_{NDO_i} + \gamma .\ Drop_{NDO_i} \qquad (2)$$

where α, β, and γ are tuning parameters specified based on the user group at requesting time of the Named Data Object (NDO), D' is the average expected delay or how long the requesting client can wait for the data, POP_{NDO} is the frequency of requesting the chunk by clients, $Drop_{NDO}$ is the probability of dropping the chunk depending on it age time-to-live (TTL).

A cache of the node implemented as FIFO queue where the first stored chunk is the first one dropped if the cache is full and the chunk's age expired. At each operational cycle, each node reset its stored chunks values ($Value_{NDO}$) and calculated them again. The simulation results showed an improvement in terms of publisher load, end-to-end delays time-to-hit data, and hit ratio when they compared it with LRU and FIFO.

The proposed strategy in [21] is a cache replacement strategy based on Content Popularity (CCP) depends on the popularity ranking of content. The authors tried to avoid the problems in LRU and LFU by adding another data structure called CPT (Content Popularity Table), a table containing all the information related to the popularity of chunks like content name, cache hit, previous and current popularity. The popularity ranking is calculated by (3) and (4).

$$P[i+1] = \frac{N[i].\alpha + P[i]}{\alpha + 1} \qquad (3)$$

$$\alpha = 1 + c.T \qquad (4)$$

Where N[i] is the number of hits of the chunk and P[i] is the last calculated popularity. The parameter α (>1) in equation (5) is proportional to counting cycle (c.T.). The influence on the number of hits will be decreasing at each time the hit occurs so the total popularity will be less affected by the time. A non-latest chunk with the lowest popularity value will be replaced when the cache is full. They simulated the strategy and compared it with LRU and LFU. In the evaluation, they focused on the average network throughput, cache hit ratio and server load. The evaluation result was evident that CCP performance is better than LRU and LFU.

The strategy used in Distributed Caching with Coordination (DCC) [23] solution is reducing the traffic on NDN network backbone by calculating the weighted popularity in two steps; one for inter-domain and the other for intra-domain. The first step, calculate Weighted Popularity (WP) for each chunk i in every node j. WP_{ij} calculated as the following:

$$\mathrm{WP}_{ij} = \frac{\mathrm{ReqCnt}_i}{\mathrm{Rank}_{ij}^{\alpha}} \qquad (5)$$

where, ReqCnt_i is a total request amount for all content in node i and $\mathrm{Rank}^{\alpha}_{ij}$ is the number of requests of one content j and a real-time skewness factor α. After obtaining the sum of all WP values in all nodes for each chunk.

When a node receives a request, it looks up to the global content dictionary. If there is a hit, it will fetch the chunk, otherwise, it will forward it to the corresponding node. The strategy is simulated and compared its Traffic-Saving Rate (TSR) and cache hit rate versus Random cache and LCE. The results showed an increase in the cache hit rate and decrease in the backbone traffic.

In [24], a proposed replacement strategy called Least Frequent Recently Used (LFRU). It is a suitable technique for rapidly changing caches. It divided the cache into two parts:

- The privileged partitions: using LRU replacement strategy. It is divided the cache into K sub-partitions. Each sub-partition has a counter for the number of hits within a Time Window (WT).
- Unprivileged partition: using an approximated LFU (ALFU). This partition should be small enough to effectively count chunks and large enough to increase the cache hit probability.

The replacement decision in this strategy depends on the condition that for each WT, it checks:

a. The request arrival rate of a chunk c_i should be higher than or equal to the minimum normalized request rate of chunks in the unprivileged partition.
b. The chunk c_i has a higher priority than the replacement candidate chunks as:

$$P(c_i) = \frac{|c_i|}{n_j} . \frac{\sum N_{ALFU}}{\tau_j(i)} \qquad (6)$$

where, c_i is the ith rank chunk, N_{ALFU} is a set of counter values of each chunk in the unprivileged partition, τ_j is the request rate of a newly arrived chunk at the jth cache node, and n_j is the size of the jth cache node that measured by the number of content items that can be stored. If the request arrival rate for the chunk is higher than the minimum normalized request rate and lower than the maximum normalized request rate of the unprivileged partition chunks, then it will drop the minimum normalized request rate chunk c_{min} from the unprivileged partition and insert c_i in the unprivileged partition. Otherwise, it will move the least recently used chunk from the selected privileged partition to the unprivileged partition. After simulating their strategy in a scale-free network generated by Barabási_Albert (BA) model using MATLAB, the results showed LFRU hit rate outperforms the Random and LRU strategies. The hit rate in LFRU is close to the window-based LFU (WLFU) and LFU.

Table 1 shows a summary of the replacement strategies discussed in this literature review. The complexity of the general replacement strategies FIFO, LRU, and RR is O(1), while LFU is O(n). The strategies FIFO, LRU, and LFU keeps the chunk list in an ordered manner while RR needs no order. All the surveyed strategies require ordered lists of chinks except UC.

Table 1. Summary of replacement strategies.

Replacement Strategy	**Complexity**	**Replaced chunk**	**Evaluation metrics**	**Parameters**
UC [4]	O(n)	Lowest CM value	Traffic-Saving Rate (TSR), cache hit rate, data usage, hit probability, number of cache units, cache size	distance, hit frequency, reachability, and priority.
LVF [22]	O(n)	Lowest	average network delay, time to hit ratio, hit ratio, publisher load	delay, hit frequency, and age. Topology is random graph
CCP [21]	O(n)	Lowest popularity	server average load, network average throughput, cache hit ratio, cache size	Last popularity, number of hits
DDC [23]	O(n)	Lowest SWP	cache size, Traffic-Saving Rate (TSR), cache hit rate, impact of request patterns, impact of content population	Request amount, and distance. Topology is Abilene and GEANT
LFRU [24]	O(1)	Minimum normalized request rate content c_{min}	probability of hits	cache size, partition rank, priority, arrival rate

4. PROPOSED METHOD

Most of the mentioned previously strategies focusing on the popularity of the chunks. Not always caching the chunks that have the highest number of requests is the best solution. Other parameters need to be considered, such as Time To Live (TTL) of chunks, distance from the requesting node, network traffic, or the cache size. Some or all mentioned parameters are selected based on network topology or specific application domain. TTL is a vital parameter for IoT device's data freshness. The proposed strategy in this paper is an enhanced LFU. In LFU, each node keeps counting the number of requests for each chunk. The cache store will be sorted in descending order where the last chunk is the least frequently used (we call it LFU), and the first chunk is the most frequently used (we call it MFU). LFU chunk has the least number of requests. LFU chunk will be replaced when the cache is full if a new chunk arrives. The proposed improvement here counts time cycles from the last request of the chunk until another request is coming to the node. The proposed strategy consists of sorting the cache and calculating the weighted popularity of a chunk.

a) Cache Sorting

Cache order kept in descending order according to chunks popularity value P. MFU chunk is a chunk with the highest value of P. At the same time, LFU is the lowest value of P. If there are two chunks with the same P value, then the recent one is stored after the second one. For example, if two chunks both have P=50, the chunk that arrived at the 10th second will be stored before the chunk arrived at the 13th second, as shown in Figure 2, where P is the calculated popularity and ToA is the time of arrival.

P:10 ToA: 32	P:30 ToA: 20	P:50 ToA: 13	P:50 ToA: 10	P:95 ToA: 2	P:100 ToA: 1

Figure 2. An example of chunks order in the cache

b) Weighted Popularity Calculation

Each chunk in the cache storage has some properties. Among these properties is the time of the last hit called Hit Time (HT), number of hits (H), and the popularity value P. Every time a request came for a stored chunk in the node, H incremented by one and HT will be sitting to the time of request arrival. After recording H and HT values, the popularity P of the chunk calculated using the following formula:

$$P_{i+1} = P_i + \frac{H_{i+1}}{HT_{i+1}} \qquad (7)$$

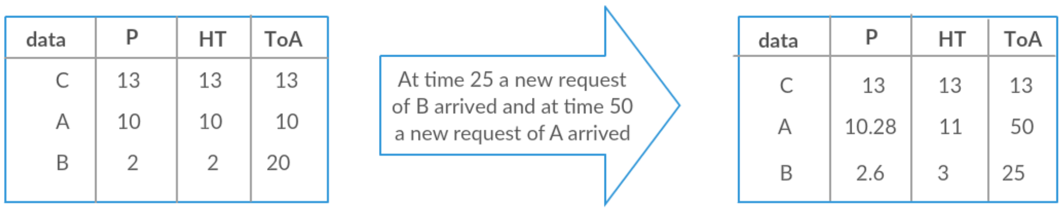
where P_i is the last calculated weighted popularity, and P_{i+1} is an update of the chunk weighted popularity. HT_{i+1} is the time from the last request arrival to the new request, expressed by the following equation:

$$HT_{i+1} = current\,time - HT_i \qquad (8)$$

where P_{i+1} = the new (updated) popularity value of the cached chunk,
P_i = the current popularity value of the cached chunk,
HT_i = the time between the last two hits,
H_i = the current number of hits;

The division by HT will affect P's value, it will be less increasing if HT is too large and the chunk becomes unpopular. When HT is small, the chunk has rapid requests, and becoming more popular, so its P value is increasing faster than other chunks.

For example, as shown in Figure 3, the chunk A has a value of P=10, the last request arrived at the second 10, and a new request arrived at the second 50, so the time between the two requests is too large HT=40, so P incremented by 0.28. On the other hand, B has a value of P=2, the last request arrived at the second 20, and a new request arrived at second 25 so HT=5 and the value of P incremented by 0.6, which is larger than the P increment of A.



data	P	HT	ToA
C	13	13	13
A	10	10	10
B	2	2	20

At time 25 a new request of B arrived and at time 50 a new request of A arrived

data	P	HT	ToA
C	13	13	13
A	10.28	11	50
B	2.6	3	25

Figure 3. Example of popularity (P) calculation

When a data chunk reaches a node, the Event-based freshness algorithm (Algorithm 1) starts invocation. If the node decided to store the chunk in the cache according to the network caching decision strategy, then the node will check if the chunk is existing in the cache or not. If it is existing, then it will increment its number of hits H by one and update its value of P. The position of the chunk will be changed depending on the new value of P. If the chunk is not existing in the cache and the cache is not full yet. It is stored at the right position at the top of the cache if it has the highest P value, at the bottom of the cache if it has the lowest value, or between two nodes where the prior node has less or equal P value and the next node has higher P value. If the cache is full, then the LFU chunk will be deleted. The new value of LFU will be the second least value of P or the newly cached chunk if it has the lowest P value. The proposed strategy is an improvement of LFU strategy. The following algorithm explains the proposed method.

Algorithm 1 Event-based freshness

```
Data: Data to retrieve
Cache Size = B
Data chunk = C
Current time = T
Number of hits = H
Last calculated popularity = P
Most frequently used chunk = MFU
Least frequently used chunk = LFU

Result:

A set of ordered chunks according to P
For each incoming data chunk decided to be stored

    if C ∈ cache
        H++
        HT = T – H
        P = P + (H / HT)
        if P > MFU→P
            Move C to the top
            MFU = C
        else if P < LFU→P
            Move C to the bottom
            LFU = C
        else if P > LFU and P < MFU
            Move C in the right position
            HT = T
        end if
    else
        if B = 0
            Place C at the top
            MFU = LFU = P
            B++
        else
            if B is full
                Delete LFU
                LFU = LFU→old
            end if
            if P > MFU→P
                Place C on the top
                MFU = C
            else if P < LFU→P
                Place C on the bottom
                LFU = C
            else if P >LFU and P < MFU
                Place C in the right position
                HT = T
            B++
            end if
        end if
```

5. EXPERIMENT RESULTS

5.1. Simulation Setup

In this section, we present detailed performance evaluation of the proposed strategy, ETFCR. We simulated GEANT network topology using ccnSim simulator with similar simulation parameters as in [12]. The ccnSim is a simulator developed using C++ to extend OMNET++ framework for NDN network simulation at data chunk level. Table 2 shows the simulation parameters used in all experiments.

Table 2. Simulation parameters.

Parameter	**Meaning**	**Values**
C	Cache size	10^2
F	File size	1 chunk
N	Number of nodes	22
Cons	Number of consumers	10
Req	Number of requests	10^2
λ	Aggregate request rate	20 req/s
R	Replicas	1
FS	Forwarding strategy	SPR (Shortest Path Routing)
Simulation_time	Simulation time	400s

5.2. Experiment Evaluation

We simulated two caching decisions: Leave Copy Everywhere (LCE) and the Probability-based caching (ProbCache). LCE is an approach that leaves a copy of the requested data in the content store of every router along the path towards the consumer. In ProbCache, the caching process is executed with a varying probability inversely proportional to the distance between the consumer and the producer. presents an unequal resource allocation among nodes, a high computational overhead, and requires parameters fine-tuning. The proposed replacement strategy (ETFCR) is evaluated with LRU, LFU, and CCP. Evaluation metrics are the average hit ratio (p_hit), the average number of hops (distance), average download time (avg_time), average network throughput, and server load.

- p_hit: it measures the average hit rate among all caches. Cache hit ratio is a measurement of how many requests a cache can fill successfully, compared to how many requests it receives. It is calculated by:

$$p\_hit = \frac{numberof\ hits}{numberof\ hits + numberof\ misses} \quad (9)$$

- hdistance: it measures the average of the number of hops for retrieving each chunk. We measure the average reduction in hop count between cached content and the original storage location. It is calculated by:

$$\text{hdistance} = \frac{\text{global average distance}}{\text{number of clients}} \quad (10)$$

- avg_time: it measures the average download time, calculated as:

$$\text{avg\_time} = \frac{\text{global average time}}{\text{number of clients}} \quad (11)$$

We extended the ccnSim simulator with ETFCR and evaluated the effectiveness of every replacement strategy. These performance metrics quantify the effectiveness of ETFCR compared with other strategies. We assume data chunk interests are at constant average rates with a randomized time gap between two consecutive Interests. A time gap is a random number that follows a uniform distribution as in [21]. The cache size varies between 20 to 400Kbits, and the bandwidth fixed at 100Mbps. Figure 4 shows the comparison results of the four caching strategies in terms of the average cache hit ratio. It shows that ETFCR has a higher hit average cache rate than LRU, LFU, and CCP. Also, we evaluate the effect of varying cache sizes. Figure 5 shows that ETFCR performance under different cache sizes. ETFCR performs significantly better than the other three caching strategies for all cache sizes. For example, for 200 Kbits of cache size, ETFCR has a cache hit ratio of approximately 0.7, compared with CCP, LFU, and LRU.

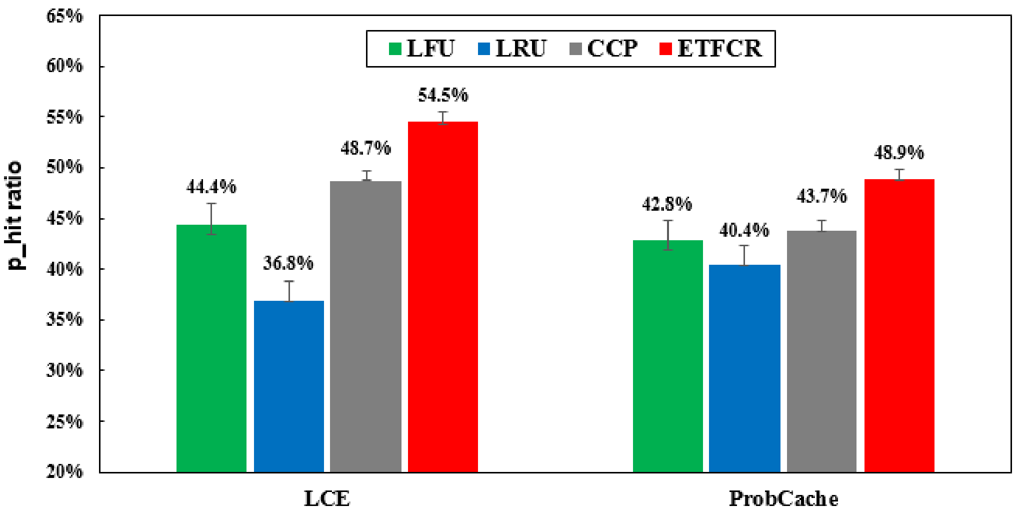


Figure 4. Evaluation of the average cache hit ratio (p_hit)

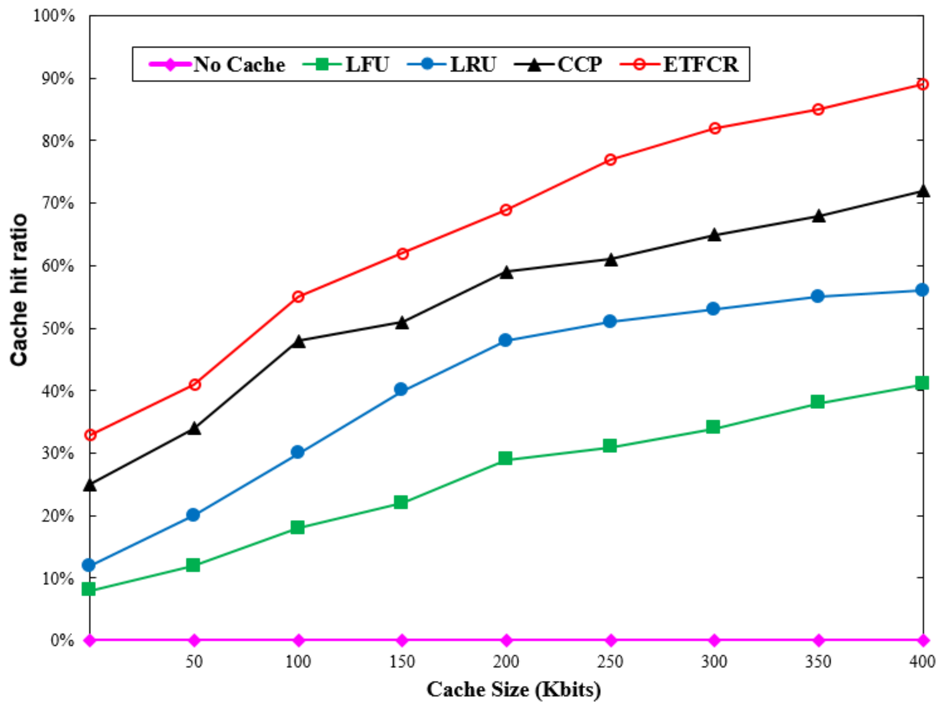


Figure 5. Cache hit ratio vs. Cache size

Figure 6 shows the difference in distance metric between ETFCR and the other three strategies. This comparison based on the total number of hops between its producer and the consumer that requested it. It shows ETFCR has good performance but with no significant difference with LRU and CCP. In Figure 7, the comparison results using avg_time show the good performance of ETFCR in the average download time compared to the remaining algorithms.

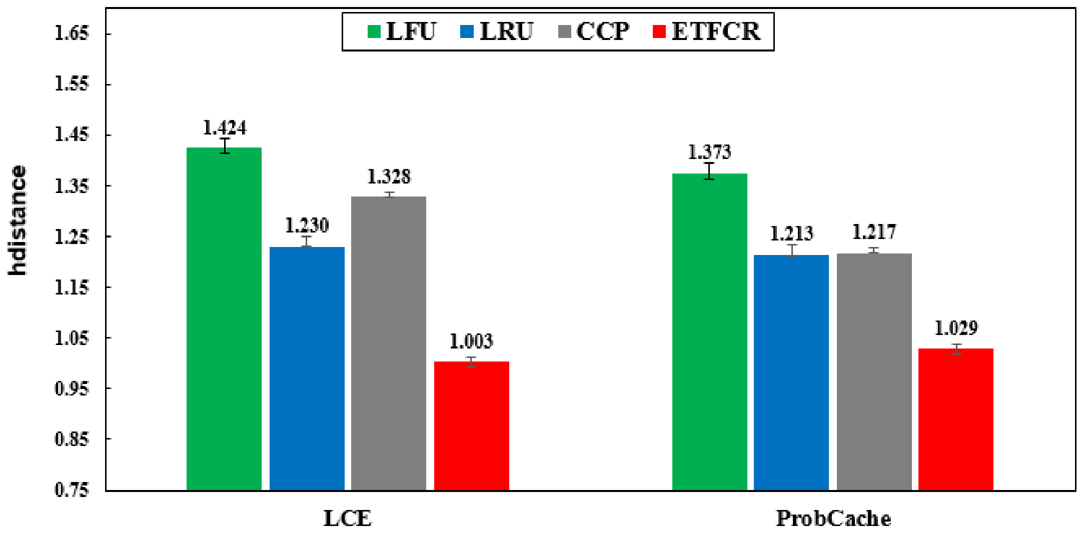


Figure 6. Evaluation of the average number of hops (hdistance)

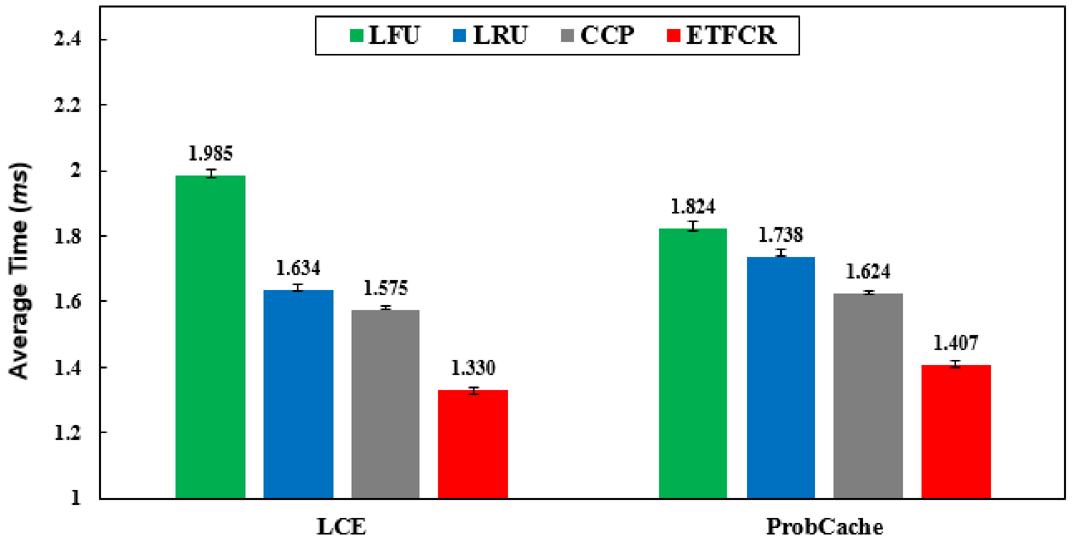


Figure 7. Evaluation of the average download time (ms)

Fig. 8 shows the performance of ETFCR in terms of average network throughput with different cache sizes where ETFCR outperforms other strategies. A Server (producer) hit occurs when an Interest could not be satisfied by any intermediate node along the path to the producer. That is when no intermediate node has a cached copy of the requested content. ETFCR has the lowest server ratio which leads to lower average server load.

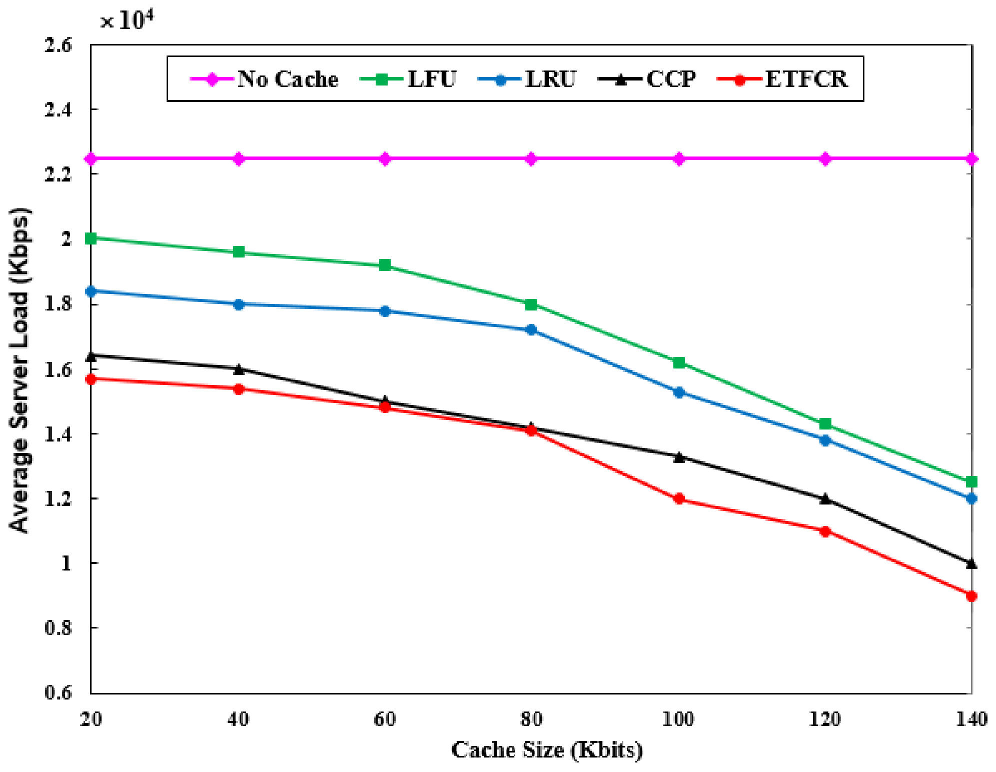


Figure 8. Evaluation of the average throughput of the network vs. Cache size

Fig. 9 shows the performance of different cache policies in the average server load with different percentages of cache capabilities. We can see the server can reach a lower load with ETFCR than with other cache policies. When the percentage of cached capacity of the node/total capacity of the network is 40%, the server load is decreased by approximately 37% compared with LFU.

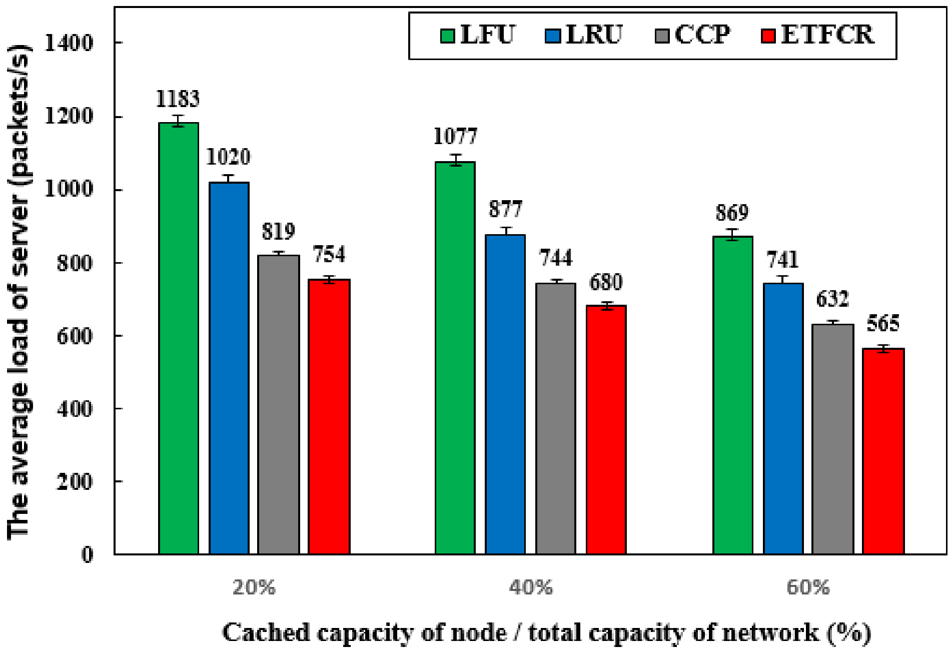


Figure 9. Evaluation of the average load of the server vs. Cache capability

6. CONCLUSIONS

NDN is one of the proposed ICN structures to replace existing location-based internet. In-network caching has a significant effect on NDN performance. There is the need to study the significance of in-networking caching in NDN from various angles. Relying on a single caching decision parameter does not give efficient caching strategies. In this paper, we proposed a new replacement algorithm called ETFCR to enhance the NDN's in-caching performance by combining multiple cache replacement factors. ETFCR enhances NDN performance by increasing the number of hits and weighted popularity. It combines request frequency and recent request time by taking into account time gaps between two successive hits. Also, we demonstrated the effectiveness of ETFCR, compared with the LRU, LFU, and CCP. The simulation results show that ETFCR increases hit rank, decreases the average distance, and the average delay time. ETFCR has the lowest server ratio which leads to lower average server load. It significantly decreases the server hot ratio with a higher consumer cache hit ratio and increases the network capacity simultaneously. Other cache replacement parameters are under consideration for future work. The selection process of these prospect parameters will depend on network application and topology.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, K. Drira, and S. Alahmadi, “Cache freshness in named data networking for the internet of things,” Comput. J., vol. 61, no. 10, 2018.

[2] L. Zhang et al., “Named data networking,” ACM SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, pp. 66–73, 2014.

[3] S. Kuribayashi, “Virtual Cache & Virtual Wan Accelerator Function Placement for Cost-Effective Content Delivery Services,” Int. J. Comput. Networks \& Commun. Vol, vol. 12, 2020.

[4] B. Panigrahi, S. Shailendra, H. K. Rath, and A. Simha, “Universal caching model and markov-based cache analysis for information centric networks,” Photonic Netw. Commun., vol. 30, no. 3, pp. 428–438, 2015.

[5] S. Hassan, R. Alubady, and A. Habbal, “Performance evaluation of the replacement policies for pending interest table,” J. Telecommun. Electron. Comput. Eng., vol. 8, no. 10, pp. 125–131, 2016.

[6] D. Saxena, V. Raychoudhury, N. Suri, C. Becker, and J. Cao, “Named Data Networking: A survey,” Comput. Sci. Rev., vol. 19, pp. 15–55, 2016.

[7] M. D. Ong, M. Chen, T. Taleb, X. Wang, and V. C. M. Leung, “FGPC: fine-grained popularity-based caching design for content centric networking,” in Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems, 2014, pp. 295–302.

[8] A. Ioannou and S. Weber, “A survey of caching policies and forwarding mechanisms in information-centric networking,” IEEE Commun. Surv. Tutorials, vol. 18, no. 4, pp. 2847–2886, 2016.

[9] B. Alahmri, S. Al-Ahmadi, and A. Belghith, “Efficient Pooling and Collaborative Cache Management for NDN/IoT Networks,” IEEE Access, vol. 9, pp. 43228–43240, 2021.

[10] M. A. Naeem, M. A. U. Rehman, R. Ullah, and B.-S. Kim, “A Comparative Performance Analysis of Popularity-Based Caching Strategies in Named Data Networking,” IEEE Access, vol. 8, pp. 50057–50077, 2020.

[11] S. J. Taher, O. Ghazali, and S. Hassan, “A Review on Cache Replacement Strategies in Named Data Network,” J. Telecommun. Electron. Comput. Eng., vol. 10, no. 2–4, pp. 53–57, 2018.

[12] R. Chiocchetti, D. Rossi, and G. Rossini, “ccnsim: An highly scalable ccn simulator,” in 2013 IEEE International Conference on Communications (ICC), 2013, pp. 2309–2314.

[13] J. François, T. Cholez, and T. Engel, “CCN traffic optimization for IoT,” in 2013 Fourth international conference on the network of the future (NOF), 2013, pp. 1–5.

[14] H. Dai, Y. Wang, H. Wu, J. Lu, and B. Liu, “Towards line-speed and accurate on-line popularity monitoring on NDN routers,” in 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS), 2014, pp. 178–187.

[15] C. Ghali, G. Tsudik, and E. Uzun, “Needle in a haystack: Mitigating content poisoning in named-data networking,” in Proceedings of NDSS Workshop on Security of Emerging Networking Technologies (SENT), 2014.

[16] T.-A. Le, N. D. Thai, and P. L. Vo, “The performance of caching strategies in content centric networking,” in 2017 international conference on information networking (ICOIN), 2017, pp. 628–632.

[17] D. Gupta, S. Rani, S. H. Ahmed, and R. Hussain, “Caching Policies in NDN-IoT Architecture,” in Integration of WSN and IoT for Smart Cities, S. Rani, R. Maheswar, G. R. Kanagachidambaresan, and P. Jayarajan, Eds. Cham: Springer International Publishing, 2020, pp. 43–64.

[18] S. Verma, R. S. Tomar, B. K. Chaurasia, V. Singh, and J. Abawajy, Communication, Networks and Computing: First International Conference, CNC 2018, Gwalior, India, March 22-24, 2018, Revised Selected Papers, vol. 839. Springer, 2018.

[19] D. Saxena, V. Raychoudhury, N. Suri, C. Becker, and J. Cao, “Named data networking: a survey,” Comput. Sci. Rev., vol. 19, pp. 15–55, 2016.

[20] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, “How to cache in ICN-based IoT environments?,” in 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), 2017, pp. 1117–1124.

[21] J. Ran, N. Lv, D. Zhang, Y. Ma, and Z. Xie, “On performance of cache policies in named data networking,” in 2013 International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013), 2013.

[22] F. M. Al-Turjman, A. E. Al-Fagih, and H. S. Hassanein, “A value-based cache replacement approach for information-centric networks,” in 38th Annual IEEE Conference on Local Computer Networks-Workshops, 2013, pp. 874–881.

[23] H. Wu, J. Li, T. Pan, and B. Liu, “A novel caching scheme for the backbone of Named data networking,” in 2013 IEEE International Conference on Communications (ICC), 2013, pp. 3634–3638.

[24] M. Bilal and S.-G. Kang, “A cache management scheme for efficient content eviction and replication in cache networks,” IEEE Access, vol. 5, pp. 1692–1701, 2017.

AUTHOR

SAAD AL-AHMADI currently an Associate Professor of computer science with the College of Computer and Information Sciences, King Saud University. He has published many papers in many journals and conferences. His research interests include cybersecurity, computer networks, mobile ad hoc networks, and sensors networks.