Article

Content Management Based on Content Popularity Ranking in Information-Centric Networks

Nazib Abdun Nasir and Seong-Ho Jeong *

Department of Information and Communications Engineering, Hankuk University of Foreign Studies, 81, Yongin-si 17035, Korea; nazib@hufs.ac.kr

* Correspondence: shjeong@hufs.ac.kr; Tel.: +82-31-330-4642

Abstract: Users can access the Internet anywhere they go at any time due to the advancement of communications and networking technologies. The number of users and connected devices are rapidly increasing, and various forms of content are becoming increasingly available on the Internet. Consequently, several research ideas have emerged regarding the storage policy for the enormous amount of content, and procedures to remove existing content due to the lack of storage space have also been discussed. Many of the proposals related to content caching offer to identify the popularity of certain content and hold the popular content in a repository as long as possible. Although the host-based Internet has been serving its users for a long time, managing network resources efficiently during high traffic load is problematic for the host-based Internet because locating the host with their IP address is one of the primary mechanisms behind this architecture. A more strategical networking paradigm to resolve this issue is Content-Centric Networking (CCN), a branch of the networking paradigm Information-Centric Networking (ICN) that is focused on the name of the content, and therefore can deliver the requested content efficiently, securely, and faster. However, this paradigm has relatively simple content caching and content removal mechanisms, as it caches all the relevant content at all the nodes and removes the content based on the access time only when there is a lack of space. In this paper, we propose content popularity ranking (CPR) mechanism, content caching scheme, and content removal scheme. The proposed schemes are compared to existing caching schemes such as Leave Copy Everywhere (LCE) and Leave Copy Down (LCD) in terms of the Average Hop Count, content removal schemes such as Least Recently Used (LRU) and Least Frequently Used (LFU) in terms of the Cache Hit Ratio, and finally, the CCN paradigm incorporating the LCE and the LRU schemes and the host-based Internet architecture in terms of Content Delivery Time. Graphical presentations of performance results utilizing the proposed schemes show that the proposed CPR-based schemes for content caching and content removal provide better performance than the host-based Internet and the original CCN utilizing LCE and LRU schemes.

Keywords: content caching; content-centric networking; content popularity ranking; content removal; information-centric networking

Citation: Nasir, N.A.; Jeong, S.-H. Content Management Based on Content Popularity Ranking in Information-Centric Networks. *Appl. Sci.* **2021**, *11*, 6088. <https://doi.org/10.3390/app11136088>

Academic Editor: C. J. Bouras

Received: 4 May 2021

Accepted: 24 June 2021

Published: 30 June 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Internet usage and data traffic loads are escalating as several applications are introduced over time. An increase in individual content sizes and diversity of content types make the network congested even more. It was predicted that by 2022, approximately 4.8 billion Internet users would be using roughly 28.5 billion connected devices, and the average Broadband Internet speed is expected to double to 75.4 Mbps by that time [1]. Consequently, the global IP traffic will increase more than 3 times to almost 400 exabytes per month from 2017 to 2022, and as much as 82% of the total IP traffic exchanged is expected to consist of video content [1]. As mobile multimedia applications are becoming popular, Internet traffic exchanges are increasing exponentially. Thus, the host-based Internet faces several challenges, such as low scalability, inadequate security, inefficient

mobility support, high bandwidth consumption, and higher latency. These disadvantages are caused primarily because the original architecture of the host-based Internet was focused on fixed machines. The IP address-based host-to-host communication mechanism creates bottlenecks and stumbles upon these drawbacks.

These downsides can be alleviated by introducing a content-centric communication system that is termed Content-Centric Networking (CCN) [2], which is a variation of the networking paradigm named Information-Centric Networking (ICN) [3]. CCN achieves flexibility in content retrieval by looking for the requested content itself and not becoming lost searching for the host that has the content. CCN provides the desired content from the nearest source by implementing an in-network caching policy that replicates all content that go through all the machines. The promising potential of CCN has prompted further research on areas including routing, security, congestion control, mobility support, and caching.

The idea of storing the requested content in advance at a closer location is called caching, and it can help deliver the content faster. If the intermediate machines cache content, the availability of that content will be increased within the network, and the content can be downloaded quickly from the intermediate machines instead of fetching it from the remote server [2]. Several policies are available that determine which content to cache and which content to discard, and the caching decision based on the popularity of the content is one of the prominent ones. This policy allows different machines to cache diverse content after considering the popularity of this content. The popularity of content is fundamentally subject to the download counter of that content, despite the fact that a few other factors need to be assessed because the popularity of the same content can be different at varying machines. The original CCN concept proposed caching all the accessed content at all the intermediate machines [2], which is inefficient due to leaving needless copies of those content and may exhaust the repository spaces. At the same time, all repositories have a fixed storage size, and as the content are continuously being generated, requested, and stored, cache overflow is just a matter of time. Therefore, when there is not enough space for caching content, an effective content removal policy needs to be executed to erase one or more existing content. Several cache replacement schemes have been proposed over time, and most of the schemes' main idea is to remove content based on the access time of that content. These schemes run the content removal procedures until there is enough free space to cache the incoming content.

Therefore, various existing schemes for content caching and cache removal face issues such as cache overflow due to inefficiently the caching content that were not necessary to be cached and cache miss due to removing the potential popular content that is requested frequently. In this paper, we propose content popularity ranking (CPR) mechanism to determine the popularity of existing content at content server and rank this content based on the calculated popularity. Furthermore, we propose content caching scheme that caches the content selectively at different machines based on the introduced CPR mechanism to reduce the cache overflow and increase the cache hit ratio. In addition, we also propose content removal scheme based on the CPR mechanism that removes content from the cache repository when there is a lack of space. The proposed scheme considers the popularity of the existing content and removes the content with the least popularity. Therefore, the probability of cache hits increases, the probability of cache misses decreases, and the content delivery times can be reduced.

The rest of the paper is organized as follows. Section 2 addresses various related work, and then Section 3 describes the proposed CPR mechanism, content caching, and content removal schemes. After that, diverse performance results are presented in Section 4, and finally, Section 5 concludes the paper.

2. Related Work

This section summarizes the related work on Information-Centric Networking, prediction of content popularity, and content caching schemes and cache removal schemes in ICN.

2.1. Information-Centric Networking

There have been various approaches based on the concept of ICN. Among them, TRIAD [4]-the first of its kind, Named Data Networking (NDN) [5], Data-Oriented Network Architecture (DONA) [6], Publish-Subscribe Internet Technology (PURSUIT) [7,8], Publish-Subscribe Internet Routing Paradigm (PSIRP) [9,10], Scalable & Adaptive Internet soLutions (SAIL) [11], Architecture and Design for the Future Internet (4WARD) [12], COntent Mediator architecture for content-aware nETworks (COMET) [13], Network of Information (NetInf) [14], and CCN [2] are worth mentioning. Despite the central idea being the same for all these variants, which is to deliver the requested content as quickly as possible using a name-based communication, there are distinctions in actual implementations in terms of routing, naming, and caching mechanisms.

CCN is one of the most popular variations that follows the ICN perception and uses the names of the requested content to locate and obtain the desired content. The basic CCN caches all the recently requested content at all the machines that fall within the transmission path for future reference. Therefore, the later requests for the same content can be satisfied with an intermediate machine rather than the remote server. Consequently, network congestion, traffic overload, and response time can be reduced significantly. In addition, CCN secures the content itself rather than the transmission link using packet-level security and supports basic user mobility. Hence, a shorter data transmission time, improved efficiency in network resource management, increased scalability concerning bandwidth demand, and better robustness in challenging communication environments are the anticipated benefits. However, further improvements are still expected in terms of content caching and content removal schemes, and mobility support.

2.2. Content Popularity Prediction

Interests in mobile multimedia applications are increasing, and the necessity of reducing the consumption of resources has become a hot topic in recent times. Therefore, content popularity prediction is an effective way to regulate the content selected for storing or removing from content repository. The generated content can be managed efficiently by identifying its popularity. Rankings of the video content can be predicted in terms of popularity by using the IMDB (Internet Movie Database) system. Additionally, news websites can identify the most viewed news and predict the types of stories their customers are mostly interested in. Predicting the popularity of various kinds of content is a well-researched area. Several content popularity methods are surveyed in [15], where limitations of different existing approaches are presented. Typically, video content consumes most of the Internet bandwidth, and in [16], models to predict daily access patterns of YouTube content are proposed using the Autoregressive Moving Average (ARMA) and Hierarchical clustering methods. However, these approaches incur an additional computational cost that may be a disadvantage. Temporal evolution prediction is used in [17] to classify various content and predict the content popularity. A Digg dataset was used to predict the popularity of news in [18]. The life duration of popular Tweets was predicted in [19] based on the static characteristics and patterns of dynamic retweeting. Placing the frequently requested content by predicting the content popularity in the ICN can improve network performance by reducing response time and traffic load of the servers. A distributed content placement strategy based on popularity for ICN was proposed in [20] that considered several aspects, including the distance between the node on the content return path and the requesting node, the content popularity trend, prediction of the future popularity of the content with the Markov Chain, and based on these, proposed to push the content with the local popular trend to the network in advance.

2.3. Content Caching and Removal Schemes

The basic CCN uses the ALWAYS caching scheme, which is the same as the Leave Copy Everywhere (LCE) scheme that means all requested content is replicated in all nodes on the transmission path. As this causes considerable cache redundancy, the caching performance was improved in [21] by incorporating the Leave Copy Down (LCD) caching policy where the content is cached only at the next-hop node from the content-access node. The Move Copy Down (MCD) caching policy was proposed to enhance the caching performance further where the original content at the content-access node is deleted after the content is cached at the next-hop node. Caching the most popular content at the nearest machine utilizing CCN was proposed by [22,23]. Then, to reduce caching costs and share the load among the nodes, Prob [24] was proposed. Similarly, ProbCache [25] introduced content caching scheme based on the remaining storage capabilities of the nodes. Other than CCN, NDN also provides a cache storage mechanism at the intermediate nodes; therefore, the concept of identifying and caching popular content works for NDN as well. Based on a compound popular content caching strategy (CPCCS), content caching mechanism was proposed for NDN in [26] that selects an optimal popular content for caching by calculating the number of requests content received. Later on, another content caching mechanism based on compound popularity was proposed for NDN in [27]. That scheme tried to increase the utilization of the existing content by considering the popularity of the content and the node popularity simultaneously. In addition, [28] proposed a new caching strategy named Most Interested Content Caching (MICC) that enhances the content distribution by caching the requested content near the consumers at various appropriate locations. Furthermore, [29] proposed another content caching scheme named Efficient Hybrid Content Placement (EHCP) to reduce the duplications of homogeneous content at several locations. This scheme also looked to increase the content diversity along the transmission path. A periodic caching strategy was proposed in [30] for the IoT environment based on NDN. The study provided simulation results in terms of content placement strategies and stretch, one of the standard performance metrics, and showed that the proposed method decreases the content retrieval time and improves the cache-hit ratio.

Besides these content caching ideas, cache replacement schemes are getting attention from researchers as well. These schemes remove one or more types of content from the repositories when there is not enough space for caching new content. The Least Recently Used (LRU) [31] scheme replaces the least recently used storage unit, which means that the content that has not been accessed for the longest time is replaced. The Least Frequently Used (LFU) [32] scheme replaces the storage unit with the least access times, meaning that the content with the least number of access requests is replaced. These two basic cache replacement schemes are simple in concept, and they are implemented regularly. Features of these two schemes are analyzed in [33]. Another basic scheme that simply replaces the oldest content from the repository is called First In First Out (FIFO). Random (RAND) [34] policy removes content randomly from the storage, which may be proved to be an inefficient policy, as popular content may be removed to store less popular content. Besides proposing a content popularity-based cache resolution scheme, a cache replacement scheme based on content age is also proposed in [35] to reduce network delay and redundancy. The pivotal concept is having a basic age and a maximum age for all the content and removing the content when the age becomes zero over time, otherwise removing the content with the lowest age if there is a lack of space. Several crucial criteria, including classifications of the content, interests of the users, the effect of the distance, feedback from caching system, and space within the storage, were considered to formulate content popularity model in [36] so that the efficiency of content distribution can be improved and the redundancy in terms of transmission for multimedia traffic can be reduced. Based on that content popularity model, new cache placement and replacement strategies were proposed using the CCN architecture.

3. Content Management Based on Content Popularity Ranking

This section describes the proposed content popularity ranking (CPR) mechanism and based on that, the proposed schemes for content caching and content removal are explained in detail.

3.1. The CPR Mechanism

The vast amount of information that needs to be stored at a specific repository is bound to become greater than the capacity of that node at one point in time or another. Additionally, as all the nodes incorporating the basic CCN concept cache all the interacted content, a considerable amount of unnecessary duplicate content is created within the network topology. Therefore, cache overflow may occur frequently, and removing existing content from the repositories becomes imperative. On the other hand, removing frequently requested content may cause cache misses in the near future. The various existing content caching schemes and content removal schemes that are used regularly may have simple algorithms to be implemented but fail to consider the potential future popularity of the content. Therefore, performance degradation may occur as the content that is going to be requested by the users may not have been cached or may be removed from the repository. Hence, to resolve these issues, we propose a content popularity ranking (CPR) mechanism that ranks the incoming content among the available content at the local storage of a server machine, and based on that, we propose new schemes for content caching and content removal. To test and evaluate our proposed CPR mechanism and the proposed schemes, we created various network topologies using NS-3 [37]. The experimental architecture consists of several machines that act as either servers or clients. The primary purpose of the client machines is to request random content from the servers repeatedly. The server machines have several objectives, including storing various content, fetching the desired content from another server when the requested content is not readily available at that server, keeping the fetched content in storage for future use, delivering the requested content to the client machines, and generating diverse experimental results on the whole process. Additionally, these servers can act as intermediary machines between the remote content servers and the requesting clients to complete the route. All these machines that are part of the content request and retrieval procedure, including the servers, the clients, and all the intermediate machines, are referred to as “nodes” hereafter.

Content is considered to be popular when several users express their interest in that particular content. However, deciding the popularity of content based on only the download counter is a long-term procedure. Therefore, several other factors should be considered to predict the popularity of content in the beginning, and the available content at the local repository of each server should be ranked in terms of its popularity at the initial stage. Moreover, the same content may have different popularity at various nodes, and the popularity of content may change over time. Thus, the process of calculating the popularity of content should be a continuous one. In order to measure the popularity of content as soon as it is stored in the server repository, we introduced tags and labels in our CPR mechanism. All the content was assigned with three tags, each consisting of a label taken from several predefined labels for those tags. Below we describe the labels of each of the tags in detail. The label assigned to a tag is expressed by x hereafter.

According to the statistics in [1], people are mostly interested in video content rather than audio or text content. Moreover, text content is generally not time-sensitive, and video content consumes higher bandwidth than audio or text content while being retrieved from remote servers. Therefore, video content generally becomes more popular than non-video content and should be kept in the repositories for a longer time, as repeated requests may occur for the same video content. We have defined a *content type* tag, expressed by T , where the file types of each content are differentiated and categorized, such as *text*, *audio*, and *video*. The servers can automatically categorize the available content in terms of the content

types and assign the appropriate label to each of the content types. The labels included within this tag are shown in (1).

$$T(x) = \{video, audio, text\} \quad (1)$$

We considered the *expected lifetime* of the content to calculate the CPR, as it plays a vital role in deciding which content should be cached for a long time and which content may be erased when there is a lack of space. The *expected lifetime* of content, expressed by E , can be easily assigned when that content is generated, and over time it can be reevaluated based on the criterion given in Section 3.3. Typically, all content may be allocated a general lifetime duration by the server administrator based on the traffic load and the storage size of that server. However, some content may already be expected to become popular in the long run, such as new releases from already popular franchises, statistical articles related to hot topics at the current time, or long-lasting general guideline information on various issues. These forms of content need to be stored at the server repositories for as long time as possible because users may keep requesting these for some time. In contrast, some other content created to serve a specific purpose for a limited time, such as guidelines for online applications and daily news updates, may have a lower lifetime expectancy to start with. Therefore, we created four labels for the tag *expected lifetime*, which are *long*, *medium*, *short*, and *zero*. These labels specify how much more time the content is expected to be popular on the Internet. The label *medium* indicates the assigned general lifetime for typical content within a server repository. The labels *long* and *short* mean a higher and lower *expected lifetime* allocated explicitly to the content, although no content may be assigned any of these two labels at the beginning in a particular server repository as well. Besides, the label *medium* of content may change to *long* or *short* over time based on the criterion explained in Section 3.3, as the popularity of the content may increase or decrease. Finally, the label *zero* asserts that the content was not popular at all over some time, and it may be removed from the server repository immediately. This label is not assigned to any content initially; instead, it may be allocated later based on the condition given in Section 3.3. The labels of this tag are given in (2).

$$E(x) = \{long, medium, short, zero\} \quad (2)$$

Besides considering the remaining lifetime, the elapsed duration since certain content was published is also essential in deciding the potential popularity of that content, as even popular content loses its popularity over time and non-popular content fades away into oblivion. We argue that new content that has just become available on the Internet and stored at a server repository should be given some time before being considered for removal, even when there is a lack of space at that server repository. The rationale behind this is that the newly created content may soon become popular among the users if given time. Removing this content would mean regular cache misses, and frequent retrieval of the same content might be needed. Therefore, a *life duration* tag was created that includes three labels that essentially indicate the age of the content and the popularity of that content. All the new content automatically gets the label *fresh* and only this label from the *life duration* tag is assigned to content at the beginning. The other two labels, *current* and *stale*, are allocated to content over time after reevaluating the popularity of that content based on the criterion explained in Section 3.3. The *fresh* content is not removed from the repository over an allocated time, the *stale* content is predominantly selected for removal, and the *current* content is also considered for removal when a lack of space in the repository persists. The labels included in the *life duration* tag, expressed by D , are given in (3).

$$D(x) = \{fresh, current, stale\} \quad (3)$$

The values assigned to the labels and the weights given to the tags determine the efficiency of the calculated *content popularity ranking*, CPR, for each example of content.

Therefore, we carefully designed a model to optimize the outcome of the CPR mechanism. A cloud content server was created where we uploaded 100 various examples of content after assigning the tags and the labels appropriately. This content included all types of audio, video, and text files of various sizes. The graduate students and their family members could access the cloud server and its content, including males, females, and children of different ages. The server stored the labels of all this content against the number of times each form of content was accessed. We collected the content request and download information from the server over two months, where 10,000 total requests were registered. The dependence count for the different labels was normalized, and the content hit distribution over different labels was categorized into 3 classes. Based on the content hit count, the labels from each tag that were requested the highest number of times were assigned the value of 3, the labels that were requested the lowest number of times were assigned the value of 1, and the remaining labels of each tag, which are in between the highest and the lowest labels, were assigned the value of 2. The label *zero* from the *expected lifetime* tag was not considered for formulating the CPR, as this label means that the content does not have any popularity. Therefore, the labels *video*, *long*, and *fresh*, respectively from the tags *content type*, *expected lifetime*, and *life duration* received the value 3, the labels *audio*, *medium*, and *current*, received the value 2, and the labels *text*, *short*, and *stale*, received the value 1. This is summarized in (4).

$$\text{Values of the labels} = \begin{cases} 3, & \text{for video} \mid \text{long} \mid \text{fresh} \\ 2, & \text{for audio} \mid \text{medium} \mid \text{current} \\ 1, & \text{for text} \mid \text{short} \mid \text{stale} \end{cases} \quad (4)$$

The number of times content is accessed is also one of the most vital pieces of information for determining the popularity of that content. Therefore, we considered the *download counter*, C_i , in our CPR mechanism, which is the number of times content i is requested. Linear regression was used to optimize the weights of the three tags, which are expressed by the tunable parameters in the formula for calculating the CPR. The formula for calculating the *content popularity ranking* of content i at a node n is given in (5).

$$CPR_i = C_{n_i} \times (\alpha \times T_i(x) + \beta \times E_i(x) + \gamma \times D_i(x)) \quad (5)$$

Here, α , β , and γ are the tunable parameters with values of 0.475, 0.325, and 0.2, respectively. C_{n_i} is the number of times content i is requested from node n . The value of C_{n_i} is generally 1 for the new incoming content, although it can be more than 1 if the same content was requested before and was not cached. However, this value is variable for the other existing content at a node, and it can vary for the same content cached at different nodes. Therefore, the same content can have different CPR at different nodes, and the popularity will also vary.

3.2. Content Caching Scheme Using the CPR Mechanism

Each node ranks all the available content at the local storage by measuring the CPR of each content using Equation (2). The objectives of the proposed caching scheme are to scatter the requested content among the surrounding nodes in such a way that the popular content, which mainly are the repeatedly requested content by the clients, are cached at the nearer nodes to the clients and stored for a longer time as well. Different server nodes may have different traffic loads and varied storage sizes. Therefore, all the content going through an intermediate node should not be cached at every node. Rather, a threshold needs to set so the content meeting the criteria of the threshold of a node can be cached at that node, and the other content which does not satisfy the criteria of the threshold of that node can be discarded. We defined a variable named *popularity threshold*, P_{TH} , which indicates the ranking position in terms of content popularity an incoming content needs to have among the existing content at a node to be cached. Every node separately calculates

its P_{TH} variable each time it needs to cache a new content. The formula for determining the *popularity threshold*, P_{TH_n} , of a node n assuming it has a total of m content is given in (6).

$$P_{TH_n} = \frac{1}{10 \times m} \sum_{j=1}^m (CPR_j) \quad (6)$$

Here, P_{TH_n} is the *popularity threshold* of a node n that has a total of m content. At first, the CPRs of all the m content are calculated, summed, and averaged so that the average CPR of all the content at that node is known. After that, the P_{TH_n} is set at 10% of the average CPR of all the content at each node. The responsible server nodes that initially receive the request for the content from the client nodes always cache those forms of content after fetching them from other server nodes. These servers move to the content removal scheme explained in Section 3.4 when there is not enough space for caching the content. On the other hand, the intermediate nodes follow Algorithm 1 and cache content only when the CPR of the content i is higher than P_{TH_n} of the node n , as shown in (7). All the intermediate nodes also use the content removal scheme explained in Section 3.4 in case there is storage space unavailability.

$$\text{Cache content } i \text{ at node } n, \text{ if } CPR_i > P_{TH_n} \quad (7)$$

By following this tactic, diversity of the cache repository can be achieved as the less popular content can be cached at the farther nodes. Additionally, network resource consumption can be optimized, and creating bottlenecks can be avoided. The procedure for content caching using the CPR mechanism is given in Algorithm 1.

Algorithm 1: cache_content ()

INPUT new content i , node n , number of total content m

1. **READ** labels of the following tags of content i
 2. $T_i(x), E_i(x), D_i(x)$
 3. **EXTRACT** value of the download counter
 4. C_{n_i} , of content i at node n
 5. **DETERMINE** values of the tags
 6. Using Equation (4)
 7. **CALCULATE** CPR of content i
 8. Using Equation (5)
 9. $CPR_i = C_{n_i} \times (\alpha \times T_i(x) + \beta \times E_i(x) + \gamma \times D_i(x))$
 10. **CALCULATE** *popularity threshold* of node n
 11. Using Equation (6)
 12. $P_{TH_n} = \frac{1}{10 \times m} \sum_{j=1}^m (CPR_j)$
 13. **IF** $CPR_i > P_{TH_n}$
 14. **CACHE** content i at node n
 15. **ELSE DISCARD** content i
 16. **END IF**
-

3.3. Updating the Labels of the Tags

The tags assigned to content are updated over time by altering the labels of the tags, excluding the *content type* tag, which is never changed for a particular content. For example, suppose content i has the labels *medium* and *fresh* for the tags *expected lifetime* and *life duration*, respectively. In that case, these labels can change to *long* or *short*, and *current* or *stale*, respectively, over time. We have introduced several new variables in order to explain the updating procedure of these labels of the tags. After caching content at the local storage, it is entitled to be stored for a duration of time without being considered for removal. This is because it is new content, and its popularity may increase in the future. This time duration variable is expressed by t_{new} . After this time, the labels *long* and *fresh* change to *medium* and *current*, respectively. After this time the duration expires, another time duration

variable starts, which is expressed by t_{life} . Several labels are updated after this time is over. These time variables indicate the duration after when the popularity of content should be recalculated. Values of these time durations are set by the administrators of the servers depending on the traffic load and the storage capacity of those servers. Afterward, another download counter variable is calculated that holds the average value of the summation of all the downloads of all the content at a node. This variable is called the *average download counter*, C_{avg} , and each node n with total m content can calculate this variable using (8).

$$C_{n_{avg}} = \frac{1}{m} \sum_{i=1}^m (C_{n_i}) \quad (8)$$

In this equation, C_{n_i} is the download counter, which is the number of times that content i is downloaded from node n . The download counters of all the available content at a node n are averaged after summing them up, and that is the value of the average download counter, $C_{n_{avg}}$, at the node n . Several of the labels assigned to content i of node n are updated after comparing the C_{n_i} value of content i with the $C_{n_{avg}}$ value of node n over the time t_{life} . If C_{n_i} gets higher than $C_{n_{avg}}$ during the time t_{life} , existing labels *medium*, *short*, and *stale* change to the labels *long*, *medium*, and *current*, respectively. On the other hand, if C_{n_i} becomes less than $C_{n_{avg}}$ within the time t_{life} , existing labels *medium* and *current* change to labels *short* and *stale*, respectively. There is no change in any other labels under these two conditions. Additionally, if C_{n_i} stays at 0 over the time t_{life} , which means that the content was not requested at all, the existing label *short* changes to the label *zero*. The label *zero* from the tag *expected lifetime* indicates the loss of interest in that content from the clients over the time t_{life} . Therefore, it can be removed from the storage conveniently, and new content can be cached. The overall procedure for updating the labels of the tags is given in Algorithm 2 and Algorithm 3. Time passed since content i was cached or t_{new} and t_{life} times expired, and a change of the existing labels of the content i at a node n is expressed by t_{n_i} in the algorithms. The time variable t_{n_i} is reset after t_{new} and t_{life} times are over, and the updating procedure restarts from the beginning.

Algorithm 2: initialize ()

INPUT existing content i , node n , number of total content m

1. **READ** labels of the following tags of content i
 2. $E_i(x), D_i(x)$
 3. **EXTRACT** value of the *download counter*
 4. C_{n_i} , of content i at node n
 5. **SET** the following time variables
 6. t_{new}, t_{life}
 7. **INITIALIZE** the following time variable
 8. t_{n_i}
 9. **CALCULATE** the *average download counter* of node n
 10. Using Equation (8)
 11. $C_{n_{avg}} = \frac{1}{m} \sum_{i=1}^m (C_{n_i})$
 12. **CALL** update_labels ()
-

Algorithm 3: update_labels ()

```

1.   IF  $t_{n_i} = t_{new}$ 
2.        $E_{long} \rightarrow E_{medium}$ ;
3.        $D_{fresh} \rightarrow D_{current}$ ;
4.   ELSE IF  $t_{n_i} = t_{life}$ 
5.       IF  $C_{n_i} \geq C_{n_{avg}}$ 
6.            $E_{medium} \rightarrow E_{long}$ ;
7.            $E_{short} \rightarrow E_{medium}$ ;
8.            $D_{stale} \rightarrow D_{current}$ ;
9.       ELSE IF  $C_{n_i} < C_{n_{avg}}$ 
10.           $E_{medium} \rightarrow E_{short}$ ;
11.           $D_{current} \rightarrow D_{stale}$ ;
12.      ELSE IF  $C_{n_i} = 0$ 
13.           $E_{short} \rightarrow E_{zero}$ ;
14.          CALL delete_content ( $i, n$ )
15.      END IF
16.  END IF
17.  CALL initialize ()

```

3.4. Content Removal Scheme Using CPR Mechanism

The proposed content caching scheme ensures that the content storage capacity of a node is handled efficiently by selectively caching the popular content, and content diversity is maintained among the neighboring nodes. However, cache overflow may still occur due to the size limitation of the repository as more and more forms of content are being generated and requested. An inefficient cache removal scheme may select the content for the replacement, which may be requested again soon. Therefore, cache misses may occur, and these forms of content would need to be fetched again from other servers. Subsequently, network resources will be consumed, and content delivery times will increase. To avoid frequent cache misses and reduce the content delivery times as much as possible, we proposed a content removal scheme using the CPR mechanism. The proposed content removal scheme selects an existing content for replacement based on the labels of the tags and the content popularity ranking in order to create enough space for incoming content that needs to be cached when there is a lack of storage capacity. The proposed content removal scheme follows the steps given in Algorithm 4 and Algorithm 5. Algorithm 4 is executed when the cache removal scheme is invoked by Algorithm 3 when content i from a node n is assigned the label *zero* for the tag *expected lifetime*, and the content i is removed from the repository of node n immediately. Algorithm 5 is executed in two scenarios: one is when an intermediate node n decides to cache a requested content i because the CPR of that content is higher than P_{TH_n} , but there is not enough space in the repository; therefore, the intermediate node has to select an existing content z for the removal. The other scenario is like this; a server node n initially received a request for content i , but the content i was unavailable in the local storage of the node n , and therefore, it was fetched from another server and then delivered to the client; now the server n is going to cache the fetched content i , but there is a lack of space in the storage; so, an existing content z has to be selected for the removal from the server node n . This process is repeated until there is enough space in the repository of node n for caching the new content i . Algorithm 4 and Algorithm 5 are given below.

Algorithm 4: delete_content (content, node)

```

INPUT existing content  $i$ , node  $n$ 
1.    $E_i(x) \rightarrow E_{zero}$ 
2.   ERASE content  $i$  from node  $n$ 

```

Algorithm 5: remove_content ()

```

INPUT new content  $i$ , existing content  $z$ , node  $n$ 
INPUT popularity threshold of node  $n$ ,  $P_{TH_n}$ 
INPUT CPR of all existing content  $z$ ,  $CPR_z$ 
1.  READ labels of the following tags of all existing content  $z$ 
2.       $E_z(x)$ ,  $D_z(x)$ 
3.  IF  $E_z(x) = E_{short}$  &&  $D_z(x) = D_{stale}$ 
4.      ERASE ALL content  $z$  from node  $n$ 
5.      IF sufficient space to cache content  $i$ 
6.          BREAK
7.      END IF
8.  ELSE IF  $E_z(x) = E_{short}$  &&  $D_z(x) = D_{current}$ 
9.      WHILE  $CPR_z < P_{TH_n}$ 
10.         ERASE ALL content  $z$  from node  $n$ 
11.      END WHILE
12.      IF sufficient space to cache content  $i$ 
13.          BREAK
14.      END IF
15. ELSE IF  $E_z(x) = E_{medium}$  &&  $D_z(x) = D_{current}$ 
16.     SORT all matching content in terms of  $CPR_z$ 
17.         lowest to highest
18.     WHILE content  $z$  exists matching this condition
19.         IF  $CPR_z < P_{TH_n}$ 
20.             ERASE content  $z$  from node  $n$ 
21.         END IF
22.         IF sufficient space to cache content  $i$ 
23.             BREAK
24.         END IF
25.     END WHILE
26. ELSE SORT all the remaining content in terms of  $CPR_z$ 
27.     lowest to highest
28.     WHILE not enough space to cache content  $i$ 
29.         ERASE content  $z$  from node  $n$ 
30.         IF sufficient space to cache content  $i$ 
31.             BREAK
32.         END IF
33.     END WHILE
34. END IF

```

The selection procedure of an appropriate content z for the removal is executed in several steps. At first, the current labels of the tags' *expected lifetime* and *life duration* are extracted from all the existing content. Then, in the 1st step, all the content with both the labels *short* and *stale* for the tags mentioned above are removed from the repository altogether. These forms of content are already near the end of their life and have less popularity. Therefore, removing all this content should not cause any disadvantages such as cache misses in the near future. After removing all of this content, if there is enough space for caching the new content i , the server nodes cache the content and break out of Algorithm 5. However, it is most likely that there may be very few forms of content, and even none may exist. In that case, if there is not enough space for storing the new content i , the algorithm goes to the 2nd step. In this step, the content with the label *short* for the *expected lifetime* tag but also with the label *current* for the *life duration* tag are selected. Then, all the content z that has both these labels and with an individual content popularity ranking CPR_z below the popularity threshold P_{TH_n} of the node n are removed together. Similar to in the previous step, if the servers can cache the content i after removing this existing content, they break out of Algorithm 5; otherwise, the algorithm continues and goes to step 3. In the 3rd step, there should be no content remaining with the label *short* for the *expected lifetime* tag, excluding the content which has the label *fresh* for the *life duration*

tag, and this should not be removed before the time t_{new} expires. Therefore, the content with the label *medium* for the *expected lifetime* tag and the label *current* for the *life duration* tag are selected in this step. It is ensured by Algorithm 3 during the updating procedure of these labels that the scenario where content has the label *medium* for the *expected lifetime* tag but the label *stale* for the *life duration* tag would never occur. Similar to the 2nd step, the content with a higher CPR_z than P_{TH_n} is excluded in this 3rd step as well. However, unlike the 2nd step, in the 3rd step, the matching content is sorted from lowest to highest according to the *content popularity ranking*, CPR_z , and only the content with the lowest CPR_z is selected for the removal. Then, Algorithm 5 checks whether there is enough space for caching the new content i or not. If there is sufficient space, then it breaks out of the content removal algorithm. If there is not sufficient space, it repeats the 3rd step of the procedures, and the next existing content z with the lowest CPR_z is removed. Therefore, in the 3rd step, the Algorithm 5 removes the selected content one by one until there is enough space for caching the new content i or any existing content remains that matches the mentioned conditions, instead of removing all the matched content together. In most circumstances, the server nodes should have enough repository space after finishing the 3rd step for caching the new content i . Nevertheless, if there is still a lack of capacity even after completing these steps, it goes to the last and final step, step 4. In the 4th step, all the remaining content z is sorted in terms of the *content popularity ranking*, CPR_z , regardless of the labels of the tags. Also, one form of content z with the lowest CPR_z is removed from the storage of node n . This process is repeated until there is adequate storage to cache the newly arrived content i . After that, Algorithm 5 completes its execution procedure, and the new content i can be stored in the server node n .

4. Performance Analysis

This Section describes the network topology, explains the experimental procedures, informs the performance measurement criteria, and demonstrates the results in graphical forms.

4.1. The Network Topology and the Experimental Procedures

The proposed schemes for content caching and cache removal were tested in several different network topologies with a random number of server and client nodes and various content in order to evaluate the performance of the schemes. The schemes were implemented using the CCN paradigm. Content request and retrieval experiments were executed in each topology by varying the X-axis parameters, including the size of the cache repository of the server nodes, the maximum number of content a client node can request to a server node, and the maximum number of clients that can attach to one server node simultaneously. Additionally, the Y-axis parameters were varied as well as mentioned in the following subsections. Therefore, the resultant graphs show the average of various outcomes from all the different topologies. For example, there were 10 server nodes in one of the network topologies, 100 client nodes of which a maximum of 12 nodes could attach to one server node at a time, and the client nodes were requesting random content from a pool of 30 different forms of content. In another network topology, a maximum of 30 client nodes could request as many as 70 forms of content to a server node, and there were 500 client nodes and 20 server nodes dispersed randomly. The cache size of different servers ranged from 10 Megabytes (MB) to a maximum of 150 MB, and the total number of available content varied from 10 forms of content and up to 90 forms of content in each experiment. Furthermore, the sizes of these content varied from 1 MB up to 100 MB. These content are taken from a total of 150 content, which consists of various video, audio, and text files, and they are all different forms of content from the ones that were available in the cloud server, which were used to measure the values of the labels. The maximum number of client nodes that can request several content from one server node started from 5 client nodes and up to 50 client nodes. The server nodes kept track of several experimental parameters that are given in the following subsections. We simulated these network topologies using NS-3 [37]. A total of 100 simulation results for each of the X-axis values were gathered and averaged

to plot the outcomes of the experiments using the graphs. Table 1 summarizes the network topology configuration.

Table 1. Configuration of the network topology.

Y-Axis Parameters		X-Axis Parameters	
Average Hop Count		Size of the Cache Repository per Server	10~150 (MB)
Cache Hit Ratio		Max Number of Forms of Content a Client can Request	10~90
Content Delivery Time		Max Number of Clients Attached per Server	5~50
Total Resources		Schemes Compared	
Nodes and Content		Caching	Removal
Servers		20	CPR-Based
Clients		100	LCE
Content		150	LCD
Network Architectures			
Host-based Internet Architecture	Basic CCN with Compared Schemes	CCN with Proposed CPR-based Schemes	

Figure 1 shows the depiction of one of the network topologies.

Figure 1. Depiction of one of the network topologies.

4.2. Average Hop Count

We assessed the Average Hop Count (AHC) as one of the key performance indicators. AHC points to the number of hops, which means the number of server nodes each content request needed to go through on average before the content could be located. The number of hops also means how many intermediate server nodes had to run the content caching scheme presented in Algorithm 1, excluding the initial server that received the content request. A lower AHC means that the request had to go fewer nodes up to find the requested content. Hence, a shorter delay in retrieving the requested content can be achieved. The value of AHC for the same content becomes less than the maximum number of hop available in the network topology as soon as the 2nd request of the same content arrives, as that content may have been cached at a nearer node. In the proposed caching scheme, the requested content is always cached at the initial server nodes, and the intermediate nodes may or may not cache the content based on the CPR of that content. When a request for

the same content comes to the same initial server node where it is already stored, it can be delivered immediately without traveling to the remote server. Although the content may be removed before the subsequent request comes, in case the storage capacity is not enough, then the content needs to be fetched again from an intermediate node or the original server node in the worst case. Therefore, the cache size plays a vital role in determining AHC, and as the cache size gets bigger, AHC becomes lower. During each content request, all the involved intermediate server nodes passed the number of hops information to the initial server nodes, and the initial server nodes calculated AHC using (9) after delivering each requested type of content to the clients.

$$AHC_n = \frac{1}{m} \sum_{i=1}^m (HopCounts_i) \quad (9)$$

Here, m is the total amount of content that the initial server node n fetched from the other servers and AHC_n is the Average Hop Count for the server node n . $HopCounts_i$ is the number of hops needed to retrieve the content i . The performance of the proposed CPR-based caching scheme was compared in terms of AHC with two other caching schemes: LCE and LCD. All these schemes were implemented separately within the CCN architecture for the experimental results of this subsection. LCE caches all the content at all the nodes; hence it should have a lower AHC; however, this content also needs to be removed frequently due to cache overflow. Therefore, the requested content needed to be fetched from other servers again, and the AHC increased as a consequence. On the other hand, LCD caches only at the next-hop node; hence it takes a longer time to store content at a server node that is nearer to the requesting client. The proposed CPR-based content caching scheme stores the frequently requested popular content at various intermediate server nodes and achieves a lower AHC by delivering the requested content from a nearer source. The results of the experiments in terms of AHC are demonstrated in Figure 2. In this graph, the maximum number of content a client can request was 40, and one server could handle a maximum of 12 clients at a time. The performance trend in terms of AHC was similar while we varied these two parameters.

Figure 2. Average hop count against increasing cache sizes.

The Y-axis indicates AHC values, and the X-axis shows the cache sizes for each of the server repositories in MB. All three content caching schemes saw a rise in performance

by achieving a decreasing value of AHC as the cache size increased. However, LCD still needed a higher number of hops than LCE, and even LCE took more hops than the proposed CPR-based caching scheme. For example, when the servers' cache size was 50 MB, servers utilizing the three different content caching schemes, CPR-based, LCE, and LCD, required 4.6, 4.95, and 5.1 hops on average, respectively, to deliver each requested form of content to the clients. As the cache size increased to 150 MB, the servers needed approximately 2, 3, and 3.5 hops on average, respectively, while utilizing the three different content caching schemes mentioned above. Therefore, it can be concluded that the CPR-based caching scheme outperformed the LCE caching scheme and the LCD caching scheme in terms of Average Hop Count, and the LCE caching scheme performed better than the LCD caching scheme.

4.3. Cache Hit Ratio

After measuring the AHC, we considered Cache Hit Ratio (CHR) as another key performance indicator. CHR is the ratio of how much of the requested content was readily available at a server node versus the total number of content requests, whether it was readily available or not, and whether it had to be fetched from the other servers. The existing content at the local storage of the server nodes needs to be replaced over time with new content as more and more content are being generated, requested, and cached. Therefore, cache misses may naturally occur due to the unavailability of the requested content at a particular server node. If the cache size of the server nodes is fixed and the number of available and requested content in a topology increases, the existing content at the local storage of a server node needs to be removed and replaced with the new content more frequently. Therefore, more cache overflows will occur, and more cache misses will follow. The Cache Hit Ratio can be increased, and the cache misses can be reduced by strategically removing the less popular content based on its CPR, as done by the proposed CPR-based cache removal scheme. A higher CHR means a better performance by the cache removal scheme, and a reduced time can be achieved to deliver the requested content. After each content request, every server node calculated its CHR using (10).

$$CHR_n = \frac{\sum_{i=1}^m (CacheHit_i)}{\sum_{i=1}^m (CacheHit_i) + \sum_{i=1}^m (CacheMiss_i)} \times 100\% \quad (10)$$

Here, m is the total number of content requests at a server node n and CHR_n is the Cache Hit Ratio of that server node. $\sum_{i=1}^m (CacheHit_i)$ is the total number of requests where the requested content was readily available at the corresponding server node n . On the other hand, $\sum_{i=1}^m (CacheMiss_i)$ is the total number of requests where the requested content was not readily available at the server node n ; hence the requested content had to be fetched from the other server nodes. The CHR was converted into a percentage figure. The performance of the proposed CPR-based cache removal scheme was compared in terms of CHR with two other cache removal schemes: LRU and LFU. As in the previous section, all these schemes were implemented separately within the CCN architecture for the experimental results of this subsection. The proposed CPR-based cache removal scheme keeps recurrently requested popular content in the repository for as long as possible and removes less popular content when there is a lack of space in the storage. However, the LRU scheme removes content that was not requested in times when there is not enough space for caching new content. By comparison, the LFU scheme removes content that was requested the lowest number of times. Both of these schemes have a higher possibility of removing content that can be requested sooner than later, as they do not consider the potential future popularity of that content. The results of the experiments in terms of CHR are demonstrated in Figure 3. In this graph, the maximum amount of available content within the network was varied, keeping the size of the cache repository fixed at 70 MB, and one server could handle a maximum of 7 clients at a time. The performance trend in terms of CHR was similar while we varied these two parameters.

Figure 3. Cache hit ratio against increasing number of content.

The Y-axis indicates CHR values, and the X-axis shows the maximum amount of available content within the network topology. The server nodes had nearly 99% CHR when the total number of available pieces of content was only 10, as the total size of the stored content almost always remained lower than the capacity of these servers. Therefore, some cache misses occurred at the beginning of the experiment, and only a few other cache misses happened during the whole experiment. As the number of content pieces increased, more cache overflows started to occur, and CHR began to decrease. Interestingly, when the maximum available content was relatively small initially, the LFU scheme performed better than the LRU scheme in terms of CHR. Servers utilizing the three different cache removal schemes, LRU, LFU, and CPR-based schemes, achieved approximately 87%, 88%, and 93% CHR, respectively, when the maximum number of available content pieces was 30. However, the LRU scheme started to perform better than the LFU scheme by successfully removing the appropriate content as the number of pieces of content increased. As a result, servers utilizing the LRU scheme achieved a higher CHR than the servers utilizing the LFU scheme. Nevertheless, servers utilizing the proposed CPR-based cache removal scheme continued to outperform both of these schemes in terms of CHR. When the maximum number of available content increased to 90, the servers utilizing the cache removal schemes based on CPR, LRU, and LFU achieved approximately 66%, 52%, and 45% CHR, respectively. Therefore, the CPR-based cache removal scheme outperformed the LFU cache removal scheme and the LRU cache removal scheme in terms of the Cache Hit Ratio, and the LRU cache removal scheme performed better than the LFU cache removal scheme when the maximum number of available content was relatively high.

4.4. Content Delivery Time

The proposed CPR-based schemes for content caching and cache removal were implemented together within the CCN architecture and evaluated in a combined manner using the key performance indicator Content Delivery Time (CDT) that shows the time elapsed from sending the content request by a client node until receiving that content completely from the server node. The previous two graphs indicated that LCE performed better than LCD as a content caching scheme, and LRU performed better than LFU as a cache removal scheme. Therefore, we combined the LCE and LRU schemes and implemented them within the CCN architecture for the experimental results of this subsection. Additionally,

the host-based Internet architecture was also brought to the spot, and content request and retrieval experiments were executed to measure the performance in terms of CDT. The differences in the content delivery times for different networks arose predominantly because of whether the requested content are readily available or not and the number of hops or distance of the server nodes from where the requested content was being delivered to. The content is always delivered from the original remote server node for the host-based Internet architecture, as no intermediate node caches the requested content. Additionally, more time is needed in order to establish a secure connection in the host-based Internet architecture. Thus, the host-based Internet architecture takes a longer time to deliver the requested content than the basic CCN architecture, which may deliver the requested content from an intermediate node and secure the content itself rather than spending time to secure the transmission path. CCN architecture incorporating LCE and LRU schemes caches all the content at all the nodes and removes the content accessed before the other content; hence, it can deliver the requested content faster than the host-based Internet from an intermediate node. However, frequent cache overflows and cache misses may occur due to the policies taken by these schemes, as explained in the previous two subsections. In contrast, server nodes utilizing the proposed CPR-based schemes for content caching and cache removal implemented within the CCN architecture can perform better than these two other architectures because of the selective caching of the frequently requested popular content and cache removal of the less popular content that ensures a smaller number of content requests have to travel farther in order to locate the requested content. The results of the experiments in terms of CDT are demonstrated in Figure 4. In this graph, the maximum number of pieces of content a client can request was 60, and the cache repository size for each server node was fixed at 130 MB. The performance trend in terms of CDT was similar while we varied these two parameters.

Figure 4. Average content delivery times against an increasing number of clients.

The Y-axis indicates CDT values, which are the required times for retrieving the content requested by the clients. The corresponding server nodes measure the CDT from the time they received content request from a client until the time that content was delivered entirely to that client. The X-axis shows the groupings of the maximum number of clients that can simultaneously attach to a server node. This number indicates the traffic load that a server node had to handle during this experiment. A larger number of clients

at a time means a higher traffic load occurs for each server node; this is the reason for the increasing trend of delay in the average CDT. The times are measured in seconds (s), and the indicated times in the Y-axis are the average times for all the clients in that group for all the successful content requests. The number of client nodes a server node can handle was increased from 5 client nodes per server node up to 50 client nodes per server node to evaluate the performance of the proposed schemes for content caching and cache removal under a high traffic load. The servers operating on the host-based Internet architecture always needed more time to deliver the requested content, were always behind the server nodes operating on the CCN-based architectures in terms of CDT, and were unable to catch up with the server nodes throughout the experiment. At the point where each server was responding to a maximum of 12 client nodes, the server nodes operating on the three different architectures: the host-based Internet architecture, the CCN architecture incorporating LCE and LRU schemes, and the CCN architecture incorporating the proposed CPR-based schemes for content caching and cache removal, required approximately 6.33 s, 5.76 s, and 3.6 s, on average, respectively, to deliver each requested content to the client nodes. The difference in performance increased even further when the traffic load increased, as each server had to handle up to 50 clients at a time. The server nodes needed approximately 24.89 s, 19.93 s, and 15.02 s on average, respectively, while operating on the three different architectures mentioned before in order to deliver each requested piece of content to the client nodes. Therefore, the content delivery times can be reduced significantly using the proposed CPR-based schemes for content caching and cache removal when the cache size of the server nodes is fixed but the traffic load is increasing.

5. Concluding Remarks

The in-network caching policy is recognized as one of the primary keys for developing fast and efficient communications and networking technologies. The schemes used for content caching and cache removal play a vital role in determining the efficiency of the in-network caching policy. An inefficient caching policy may produce unnecessary duplicates of the content, causing regular cache overflow. Thus, cache misses may occur, and delivery times for the requested content may increase. The CCN paradigm can alleviate some of the drawbacks of the host-based Internet architecture; however, the schemes used by the original CCN architecture for content caching and cache removal are simple in concept and can be enhanced. This paper proposes a content popularity ranking (CPR) mechanism, content caching scheme, and content removal scheme for the ICN-based networks. The CPR mechanism takes into consideration several aspects of the requested content and ranks them in terms of content popularity among the existing content of the server nodes. The proposed CPR mechanism and the proposed schemes for content caching and cache removal are described in detail. The main objectives of the proposed schemes were to identify the more popular content and cache them in the server nodes. The objectives were also to select the less popular content and remove them from the repositories when there is a lack of storage space. The proposed schemes were compared to existing schemes for content caching such as Leave Copy Everywhere (LCE) and Leave Copy Down (LCD) in terms of Average Hop Count, cache removal such as Least Recently Used (LRU) and Least Frequently Used (LFU) in terms of the Cache Hit Ratio, and finally, the CCN paradigm incorporating LCE and LRU schemes and the host-based Internet architecture in terms of Content Delivery Time. Graphical presentations of the performance results show that the proposed CPR-based schemes for content caching and cache removal provide better performance in terms of the mentioned performance criteria than the host-based Internet and the original CCN utilizing LCE and LRU schemes.

Author Contributions: Conceptualization, N.A.N. and S.-H.J.; methodology, N.A.N.; software, N.A.N.; validation, N.A.N. and S.-H.J.; formal analysis, N.A.N. and S.-H.J.; investigation, N.A.N. and S.-H.J.; writing—original draft preparation, N.A.N.; writing—review and editing, N.A.N. and S.-H.J.;

supervision, S.-H.J.; project administration, S.-H.J.; funding acquisition, S.-H.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was financially supported by the Ministry of Trade, Industry, and Energy (MOTIE) and Korea Institute for Advancement of Technology (KIAT) through the International Cooperative R&D program. This work was partly supported by the ICT R&D program of MSIT/IITP. This work was supported by Hankuk University of Foreign Studies Research Fund of 2020.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

This manuscript includes the following abbreviations.

AHC	Average Hop Counts
ARMA	Autoregressive Moving Average
CCN	Content-Centric Networking
CPR	Content Popularity Ranking
CDT	Content Delivery Time
COMET	Content Mediator architecture for content-aware nETworks
CHR	Cache Hit Ratio
DONA	Data-Oriented Network Architecture
EHCP	Efficient Hybrid Content Placement
FIFO	First In First Out
ICN	Information-Centric Networking
IMDB	Internet Movie Database
LCE	Leave Copy Everywhere
LCD	Leave Copy Down
LRU	Least Recently Used
LFU	Least Frequently Used
MICC	Most Interested Content Caching
MCD	Move Copy Down
MB	MegaBytes
NDN	Named-Data Networking
NetInf	Network of Information
PURSUIT	Publish-Subscribe Internet Technology
PSIRP	Publish-Subscribe Internet Routing Paradigm
RAND	Random
SAIL	Scalable & Adaptive Internet soLutions
s	Seconds
4WARD	Architecture and Design for the Future Internet

References

1. Barnett, T.; Jain, S.; Andra, U.; Khurana, T. *Cisco Visual Networking Index (VNI), Complete Forecast Update, 2017–2022*; 2018. Available online: <https://s3.amazonaws.com/media.mediapost.com/uploads/CiscoForecast.pdf> (accessed on 30 June 2021).
2. Jacobson, V.; Smetters, D.K.; Thornton, J.D.; Plass, M.F.; Briggs, N.H.; Braynard, R.L. Networking named content. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, Rome, Italy, 1–4 December 2009; pp. 1–12.
3. Xylomenos, G.; Ververidis, C.N.; Siris, V.A.; Fotiou, N.; Tsilopoulos, C.; Vasilakos, X.; Katsaros, K.V.; Polyzos, G.C. A survey of information-centric networking research. *IEEE Commun. Surv. Tutor.* **2013**, *16*, 1024–1049. [CrossRef]
4. Cheriton, D.R.; Gritter, M. TRIAD: A New Next-Generation Internet Architecture. 2000. Available online: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.5878> (accessed on 30 June 2021).
5. Zhang, L.; Afanasyev, A.; Burke, J.; Jacobson, V.; Claffy, K.C.; Crowley, P.; Papadopoulos, C.; Wang, L.; Zhang, B. Named data networking. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 66–73. [CrossRef]

6. Koponen, T.; Chawla, M.; Chun, B.G.; Ermolinskiy, A.; Kim, K.H.; Shenker, S.; Stoica, I. A data-oriented (and beyond) network architecture. In Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, 27–31 August 2007; pp. 181–192.
7. Trossen, D.; Parisi, G. Designing and realizing an information-centric internet. *IEEE Commun. Mag.* **2012**, *50*, 60–67. [\[CrossRef\]](#)
8. FP7 PURSUIT Project. Available online: <http://www.fp7-pursuit.eu/> (accessed on 21 August 2018).
9. Lagutin, D.; Visala, K.; Tarkoma, S. Publish/Subscribe for Internet: PSIRP Perspective. *Future Internet Assem.* **2010**, *84*, 75–84.
10. FP7 PSIRP Project. Available online: <http://www.psirp.org/> (accessed on 21 August 2018).
11. FP7 SAIL Project. Available online: <http://www.sail-project.eu/> (accessed on 21 August 2018).
12. FP7 4WARD Project. Available online: <http://www.4ward-project.eu/> (accessed on 21 August 2018).
13. García, G. COMET: Content mediator architecture for content-aware networks. In Proceedings of the 2011 Future Network & Mobile Summit, Warsaw, Poland, 15–17 June 2011; pp. 1–8.
14. Dannewitz, C.; Kutscher, D.; Ohlman, B.; Farrell, S.; Ahlgren, B.; Karl, H. Network of information (netinf)—An information-centric networking architecture. *Comput. Commun.* **2013**, *36*, 721–735. [\[CrossRef\]](#)
15. Nancy, J.G.A.; Kumar, D. Content popularity prediction methods—A survey. In Proceedings of the 2018 3rd International Conference on Communication and Electronics Systems (ICCES), Coimbatore, Tamil Nadu, India, 15–16 October 2018; IEEE: Piscataway, NJ, USA, 2018.
16. Gursun, G.; Crovella, M.; Matta, I. Describing and forecasting video access patterns. In Proceedings of the 2011 Proceedings IEEE INFOCOM, Shanghai, China, 15–16 April 2011; IEEE: Piscataway, NJ, USA, 2011.
17. Ahmed, M.; Spagna, S.; Huici, F.; Niccolini, S. A peek into the future: Predicting the evolution of popularity in user generated content. In Proceedings of the Sixth ACM International Conference on Web Search and Data Mining—WSDM '13, Rome, Italy, 4–8 February 2013; ACM Press: New York, NY, USA, 2013.
18. Lerman, K.; Hogg, T. Using a model of social dynamics to predict popularity of news. In Proceedings of the 19th International Conference on World Wide Web—WWW '10, Raleigh, NC, USA, 26–30 April 2010; ACM Press: New York, NY, USA, 2010.
19. Kong, S.; Feng, L.; Sun, G.; Luo, K. Predicting lifespans of popular tweets in microblog. In Proceedings of the Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval—SIGIR '12, Portland, OR, USA, 12–16 August 2012; ACM Press: New York, NY, USA, 2012.
20. Li, Y.; Li, R.; Yu, M. A distributed content placement strategy based on popularity for ICN. In Proceedings of the 2019 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), Xiamen, China, 16–18 December 2019; IEEE: Piscataway, NJ, USA, 2019.
21. Laoutaris, N.; Che, H.; Stavrakakis, I. The LCD interconnection of LRU caches and its analysis. *Perform. Eval.* **2006**, *63*, 609–634. [\[CrossRef\]](#)
22. Dabirmoghaddam, A.; Barijough, M.M.; Garcia-Luna-Aceves, J.J. Understanding optimal caching and opportunistic caching at “the edge” of information-centric networks. In Proceedings of the 1st International Conference on Information-Centric Networking—INC '14, Paris, France, 24–26 September 2014; ACM Press: New York, NY, USA, 2014.
23. Suksomboon, K.; Tarnoi, S.; Ji, Y.; Koibuchi, M.; Fukuda, K.; Abe, S.; Motonori, N.; Aoki, M.; Urushidani, S.; Yamada, S. PopCache: Cache more or less based on content popularity for information-centric networking. In Proceedings of the 38th Annual IEEE Conference on Local Computer Networks, Sydney, NSW, Australia, 21–24 October 2013; IEEE: Piscataway, NJ, USA, 2013.
24. Arianfar, S.; Nikander, P.; Ott, J. On content-centric router design and implications. In Proceedings of the Re-Architecting the Internet Workshop on—ReARCH '10, Philadelphia, PA, USA, 30 November 2010; ACM Press: New York, NY, USA, 2010.
25. Psaras, I.; Chai, W.K.; Pavlou, G. In-network cache management and resource allocation for information-centric networks. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 2920–2931. [\[CrossRef\]](#)
26. Naeem, N.; Hassan, K. Compound popular content caching strategy in named data networking. *Electronics* **2019**, *8*, 771. [\[CrossRef\]](#)
27. Gui, Y.; Chen, Y. A cache placement strategy based on compound popularity in named data networking. *IEEE Access* **2020**, *8*, 196002–196012. [\[CrossRef\]](#)
28. Naeem, M.A.; Ali, R.; Alazab, M.; Meng, Y.; Zikria, Y.B. Enabling the content dissemination through caching in the state-of-the-art sustainable information and communication technologies. *Sustain. Cities Soc.* **2020**, *61*, 102291. [\[CrossRef\]](#)
29. Meng, Y.; Naeem, M.A.; Ali, R.; Kim, B.-S. EHCP: An efficient hybrid content placement strategy in named data network caching. *IEEE Access* **2019**, *7*, 155601–155611. [\[CrossRef\]](#)
30. Naeem, M.; Ali, R.; Kim, B.-S.; Nor, S.; Hassan, S. A periodic caching strategy solution for the smart city in Information-centric Internet of Things. *Sustainability* **2018**, *10*, 2576. [\[CrossRef\]](#)
31. Dan, A.; Towsley, D. An approximate analysis of the LRU and FIFO buffer replacement schemes. In Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems—SIGMETRICS '90, Boulder, CO, USA, 22–25 May 1990; ACM Press: New York, NY, USA, 1990.
32. Lee, D.; Choi, J.; Kim, J.-H.; Noh, S.H.; Min, S.L.; Cho, Y.; Kim, C.S. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems—SIGMETRICS '99, Atlanta, GA, USA, 1–4 May 1999; ACM Press: New York, NY, USA, 1999.

33. Hasslinger, G.; Heikkinen, J.; Ntougias, K.; Hasslinger, F.; Hohlfeld, O. Optimum caching versus LRU and LFU: Comparison and combined limited look-ahead strategies. In Proceedings of the 2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), Shanghai, China, 7–11 May 2018; IEEE: Piscataway, NJ, USA, 2018.
34. Laoutaris, N.; Syntila, S.; Stavrakakis, I. Meta algorithms for hierarchical Web caches. In Proceedings of the IEEE International Conference on Performance, Computing, and Communications, Phoenix, AZ, USA, 15–17 April 2004; IEEE: Piscataway, NJ, USA, 2005.
35. Zhou, X.; Ye, Z. Popularity and age based cache scheme for content-centric network. In Proceedings of the 2017 3rd International Conference on Information Management (ICIM), Chengdu, China, 21–23 April 2017; IEEE: Piscataway, NJ, USA, 2017.
36. Wang, Q.; Zhu, X.; Ni, Y.; Gu, L.; Zhao, H.; Zhu, H. A new content popularity probability based cache placement and replacement plan in CCN. In Proceedings of the 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, 16–19 October 2019; IEEE: Piscataway, NJ, USA, 2019.
37. Ns-3 | A Discrete-Event Network Simulator for Internet Systems. Available online: <https://www.nsnam.org/> (accessed on 15 October 2014).