electronics

Article

A Cuckoo Filter-Based Name Resolution and Routing Method in Information-Centric Networking

Wenhan Lian [1,2], Yang Li [1,2], Jinlin Wang [1,2,3] and Jiali You [1,2,3,*]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
[2] School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China
[3] Peng Cheng Laboratory, Shenzhen 518055, China
* Correspondence: youjl@dsp.ac.cn

Abstract: Information-centric networking (ICN) is a new network architecture that routes content based on names to improve transmission performance. Therefore, the efficiency of name resolution and routing becomes a critical issue in ICN. The bloom filter-based routing scheme has gained significant attention for its ability to improve the memory efficiency of routing nodes in the network, but it cannot handle the movement or deletion of content and has a high false positive rate, which increases bandwidth consumption. In this paper, we propose a cuckoo filter-based name resolution and routing method where resolution requests are forwarded through a hierarchical network structure to the node closest to the content copy as much as possible to minimize latency. This method achieves reliable content removal and allows summaries of content to be exchanged between nodes for resolution error correction and information synchronization based on a modified cuckoo filter. The simulation results show that our method can effectively reduce the number of false positives, and it can reduce the additional overhead caused by processing false positives for a large-scale network by 50% compared with the bloom filter-based scheme.

Keywords: name-based routing; name resolution; cuckoo filter; information-centric networking

Citation: Lian, W.; Li, Y.; Wang, J.; You, J. A Cuckoo Filter-Based Name Resolution and Routing Method in Information-Centric Networking. *Electronics* **2022**, *11*, 3243. https://doi.org/10.3390/electronics11193243

Academic Editor: Martin Reisslein

Received: 31 August 2022
Accepted: 7 October 2022
Published: 9 October 2022

1. Introduction

Information-centric networking (ICN) is a future content-centric network architecture designed to address the shortcomings of the host-centric Internet architecture. ICN regards information as the center of communication, which is in line with the primary need of web users to access content. Several projects have been conducted in recent years for the research and development of ICN [1–6]. In order to improve transmission efficiency and content security, these projects use hierarchical or flat naming methods to name content and decouple content names from their locations at the network layer. Each content object has a unique name, and communication is based on the content name rather than the location and address of the content source so that users can still retrieve content even if the location of the content changes.

Considering the context of mobile Internet development, the number of content objects will continue to grow. The sheer volume of network content generates a huge name space, and these collections of names not only incur huge storage costs, but the amount of computation required to search for names to perform routing functions would increase significantly. At the same time, routing devices in the network only have limited storage resources and computing power, so name-based forwarding in ICN is bound to face scalability challenges.

To cope with this problem, there are currently two main types of name resolution and routing techniques: name-based routing and the name resolution service [7]. The former [8,9], represented by NDN [1] and CCN [4], uses a flooding protocol at the network

layer for forwarding based on the name in the request, which can reduce the size of the forwarding table due to the aggregation advantage of hierarchical names. In this type of scheme, name resolution and content routing are coupled, the content request is routed to the provider, and the requested content is then forwarded to the requester along the reversed path of the content request route. Name routing between nodes via flooding generates a large number of messages and results in high traffic costs. The latter [10,11], represented by MobilityFirst [2] and PURSUIT [5], sets up several resolution nodes in the network to store the relationship between names and network locations, maps the content names to the corresponding resolution node through a hash scheme, and forwards the requests to the content sources using the addresses provided by the resolution service. In this type of solution, the name resolution process is decoupled from the content routing process. The name resolution service system, using a distributed hash table (DHT), solves the problem of scalability and supports flat-name ICN, but this system is implemented entirely based on the overlay network, and it often lacks the location information of the underlying physical network, which leads to long resolution paths and long time delays.

As an effective tool to improve scalability, the bloom filter (BF) [12] has received a lot of attention, and there has been a lot of research [13–16] integrating this concept of a probabilistic data structure into routing schemes. For name-based routing [17], every routing node sets multiple BFs which can quickly exclude links that do not meet the forwarding requirements, thus greatly reducing the bandwidth consumption caused by flooding. In the bloom filter-based routing (BFR) approach [18], the required bandwidth and storage overhead for propagating and storing BFs grows linearly with the number of available content object names [19]. Recently, pull-based BFR [20] was proposed as a new interest-forwarding strategy for the typical scenario with a very large number of content objects. It only advertises the names of the requested content [21], and thus the routers do not need to store the entire content universe. The solution proposed by Lee et al. [22] is also applicable to this scenario, as they designed a dual-load bloom filter, which is basically a quarternary BF. These improvements reduce the storage pressure on the router, but there is no essential change in name routing. As for the name resolution service, the content names are stored in the BF, which can accelerate the retrieval process and improve the storage efficiency [7]. For flat ID-based ICN [14], the reasonable coding method of a BF can even realize name aggregation of the same prefix and finally allow it to achieve NDN-style forwarding.

However, the bloom filter has inherent drawbacks. It does not support deletion, which also causes the BF-based name routing schemes to experience difficulties in handling the deletion and movement of content, which happens almost all the time in mobile Internet application scenarios. In addition, the BF suffers from false positives. Once the name reaches the wrong destination in the routing process, subsequent content delivery cannot be guaranteed.

In this paper, we combine the filter and DHT to design a cuckoo filter-based name resolution and routing scheme to support flat-name ICN and ensure good scalability. We have improved on the traditional DHT-based name resolution system architecture. Outside the DHT overlay network, intra-domain name resolution and routing is performed by infrastructure routers running IP routing protocols, resulting in a hierarchical structure. These infrastructures are stable and physically located closer to the content source, and they can achieve fast request forwarding by existing intra-domain routing protocols such as OSPF, thus ensuring efficient intra-domain resolution. In addition, we study the impact of false positive matches on the correctness of packet forwarding. In fact, even though the cuckoo filter has a lower false positive rate, false forwards are inevitable when the network is large. We can effectively avoid erroneous forwarding by designing the cooperation mechanism of the nodes in our resolution system, and the high-frequency access content will not even be affected due to false positives.

The contributions of this paper are as follows:

- We compare the differences in the false positive rates between commonly used probabilistic data structures, including the bloom filter, the cuckoo filter and some variants. We propose a simplified adaptive cuckoo filter (SACF) which maintains efficient query performance with some error correction capability and performs well in multiple queries on popular content. We also apply it for the first time to name resolution and routing in ICN.
- We design a new name resolution and routing scheme based on the SACF. This hierarchical resolution scheme accommodates both flat names and hierarchical names, and it integrates name resolution into the routing and forwarding process. Requests will be forwarded to the resolution node closest to the content copy as much as possible to reduce resolution latency.
- We design a content movement support mechanism for our scheme to make content removal reliable, as well as an error correction feedback mechanism to address false positives caused by the filter, thus achieving a better balance between the efficiency and overhead of name routing.

The remainder of this article is organized as follows. In Section 2, we introduce and describe the probabilistic structures in ICN routing: the bloom filter and the cuckoo filter. In Section 3, we describe a novel cuckoo filter-based name resolution and routing method. In Section 4, we evaluate the effectiveness of our proposed method. Finally, the conclusion is presented in Section 5.

2. Related Work

In this section, we will briefly introduce the structure and characteristics of the traditional bloom filter and describe the benefits obtained by the current ICN routing scheme incorporating the bloom filter and, of course, the limitations imposed by the bloom filter. In addition, we introduce the cuckoo filter, a novel probabilistic data structure, and its variants and give the reasons why it is more suitable for the ICN routing scheme than the bloom filter.

2.1. Bloom Filter-Based Routing Approach

The bloom filter (BF) [12] is a space-saving probabilistic data structure, and it is used to determine whether a given element is a member of a set. A positive answer means that the item has a high probability of being in the set, and a negative answer means that it absolutely does not exist. The BF is essentially an array of fixed-length bits that takes up little memory, and all bits are initialized to zero. During the insertion of an element x, the BF uses a set of k hash functions $Hash_1(x)$, $Hash_2(x)$, ..., $Hash_k(x)$ to calculate the corresponding k positions and set them to one. Similarly, to look up if an element has been inserted into the BF, those positions are read, and a positive result is returned only if all of them are one. A bloom filter with several elements already inserted is shown in Figure 1.

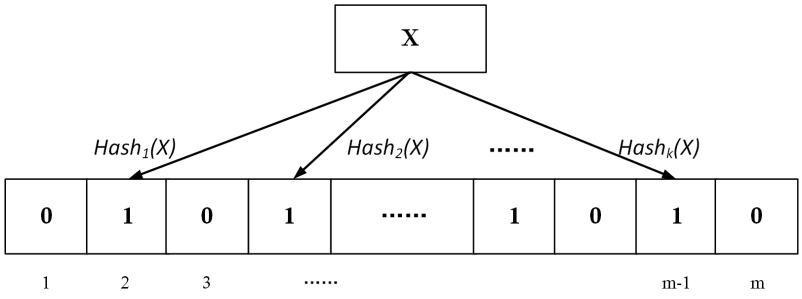

Figure 1. Illustration of a bloom filter checking an element x.

However, the design of the BF dictates that it will have a relatively low probability of false positives; that is, when checking for an element that has not been inserted into the

BF, the filter may give a positive value. This happens when the k positions being checked have been set to one when inserting other elements. The probability of a false positive for a single query in a BF can be approximated by FPR_{BF}, which is actually the probability that all k positions queried have been set to one. Suppose a BF has a size of m bits, and the probability of any one position being set to one after n elements have been inserted is p_1:

$$p_1 = 1 - (1 - \frac{1}{m})^{nk} \tag{1}$$

Therefore, the false positive rate for the BF is calculated as in Equation (2). In general, the parameter m is set to a large number by default:

$$FPR_{BF} = (1 - (1 - \frac{1}{m})^{nk})^k \approx (1 - e^{-\frac{nk}{m}})^k \tag{2}$$

To obtain the ideal false positive rate, the optimal number of hash functions is set according to Equation (3):

$$k = \frac{m}{n} ln2 \tag{3}$$

Not only does the BF suffer from false positive errors, but it also fails to support the deletion of elements. Deletion may cause false negative problems, as a particular position may already be set to one by more than one element. In order to support deletion, there are some improved variants of the BF, but this greatly increases the memory requirements. For example, the counting bloom filter (CBF) [23] uses a four-bit counter, meaning that the memory used is increased by four times. The ternary bloom filter (TBF) [24] uses three values at each position. Except for zero and one, x indicates a collision, but this filter can only support partial deletion, which also requires more space than the BF. The time complexity of the BF in retrieval is $O(1)$, and the amount of memory usage is reduced by approximately 75% by using the BF according to an existing study [25].

In P2P networks, the BF is beneficial in improving file sharing performance [26]. Chen et al. [27] used a bloom filter in an overlay based on DHT global inverted indexes, and the BF reduced the unnecessary network load when performing multi-keyword searches in P2P networks. Ariyoshi et al. [28] proposed a distributed algorithm to process conjunctive queries in P2P DHTs. When a requester issues a conjunctive query, each peer having that file conducts a search operation in its bloom filter. A bloom filter-based dual-layer scheme is an inter-domain routing scheme [29]. It is based on a two-layer BF that looks up the IP in the network gateway to implement internal and external routing layers.

With a large and growing amount of routing table entries but limited memory per node, the BF has become an important tool for solving routing problems in ICN. In the existing BF-based routing scheme [9,18], the node maintains a BF for each source or link to store the published content. The node matches the name carried in the request with all BFs it maintains to decide the next forwarding. There are also some deficiencies in these methods because of the BF. First, the inability of BF to support deletion directly results in these routing solutions not being able to deal with moved or deleted content. The authors of [18] exchanged the entire bloom filter between nodes for information synchronization, which imposed an unacceptable communication overhead, and the authors of [9] used an improved SBF to support deletion at the expense of the actual storage space efficiency. Futhermore, Katsaros et al. [30] analyzed the high false positive rate (FPR) of the BF-based routing scheme in ICN, especially in hierarchical network architectures, where the false positives accumulate as the number of hops increases and eventually affects the correctness of forwarding.

2.2. Cuckoo Filter

As an alternative to the BF, the cuckoo filter (CF) [31] is functionally very similar to the BF and has gained a lot of attention in recent years. Structurally, a CF is a two-dimensional matrix consisting of an array of m buckets, with each bucket including c slots for storing

fingerprints. It can also be interpreted as a table with m rows and c columns, but this table stores the fingerprints. Specifically, the fingerprint is actually an abstract representation of the original element. Assuming that an element is x, its fingerprint fp_x can be obtained by performing a hash operation on it; that is, $fp_x = h_f(x)$. The length of the fingerprint f depends on the size of the slot, where the larger the slot, the longer the fingerprint, and the more accurate the representation of the original element x. Because the fingerprint fp_x takes a much smaller number of bits than x itself, the CF is also a space-efficient structure. Figure 2 shows an example of a CF with m buckets, each with four slots.

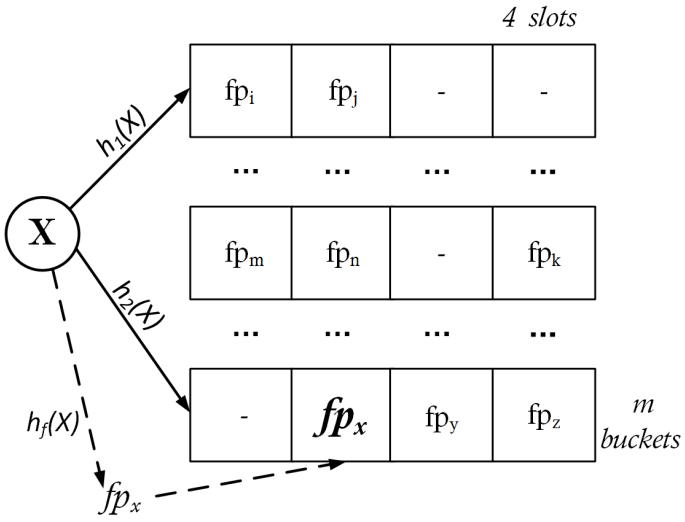

Figure 2. Illustration of a cuckoo filter checking an element x.

During insertion, each element usually has two optional candidate buckets, which are also obtained by performing a hash operation on that element. The positions of the candidate buckets are calculated as shown in Equation (4). The advantage of this design is that the location of another candidate bucket of an element can be obtained by XOR calculation between the current bucket and the fingerprint, solving the relocation problem encountered when storing the fingerprint. If the two candidate buckets corresponding to the element x are all full, then a random fingerprint is kicked out at this point, and fp_x is stored in preference. The kicked victim can then calculate the location of its other bucket and complete the insertion. This process may take a lot of time if it produces many victims, and this is where the CF is inferior to the BF:

$$\begin{cases} h_1(x) = hash(x) \\ h_2(x) = h_1(x) \oplus hash(fp_x) \end{cases} \tag{4}$$

The query process is much more efficient. It only needs to compare the fingerprint of the element to be checked with the maximum $2c$ fingerprints of the two optional buckets to complete the judgment. It should be noted that a false positive will occur when querying for an uninserted element that may happen to have the same fingerprint as in the CF. Assuming that the current load rate of CF is a, a query compares at most $2ac$ fingerprints at this time, and the probability that two fingerprints with a length f are completely identical is $\frac{1}{2^f}$. Therefore, the false positive rate of the CF can be calculated by Equation (5). It is important to note that the simplification holds when $2ac \ll 2^f$:

$$FPR_{CF} = 1 - (1 - \frac{1}{2^f})^{2ac} \approx \frac{2ac}{2^f} \tag{5}$$

Compared with a BF, the greatest advantage of a CF is the support of the delete operation. A CF achieves deletion by removing the fingerprint of an element x. Obviously, this will not have any impact on the searching of other elements. In addition, compared

with the BF, the false positive rate of the cuckoo filter is even lower, and there are many variants to further reduce the false positive rate. For example, the configurable bucket cuckoo filter (CBCF) [32] reduces hash collisions by increasing the average length of the fingerprints, and the adaptive cuckoo filter (ACF) [33] takes advantage of the one-to-one correspondence between fingerprints and elements to reduce the false positives in practice by modifying the incorrectly matched fingerprints in the query. Finally, Table 1 briefly compares some of the probabilistic data structures mentioned in this section. For reference, the FPR values are based on the experimental results of this paper.

Table 1. Comparison of several filters.

Filter	Deletion Support	FPR for Optimal Settings	Extra Cost
BF	No	Good (0.021)	No
CBF	Yes	Same as BF	Counters
TBF	Partial	Same as BF	Counters
CF	Yes	Close to BF (0.029)	No
ACF	Yes	Better for repeat queries (0.005)	Flags and a main table for elements

3. Proposed Method

In this part, we propose the cuckoo filter-based name resolution and routing method. First, we analyze the impact of bloom filter false positives on the bloom filter-based name resolution and routing scheme in Section 3.1. Then, the structure of our resolution system and the routing process for name requests are described in Section 3.2. In Section 3.3, we propose the SACF and introduce it into ICN name routing for the first time. Finally, we design two mechanisms for our system to address the negative effects of the filters in Section 3.4.

3.1. Problem Statement

The false positives of the bloom filter will have an impact on the forwarding of a router and thus affect the correctness of name routing in the network. In the following, we will explain how packets deviate from the correct forwarding path step by step.

Suppose a routing node has R different egress interfaces, and a bloom filter is set for each interface. The node has the name information required for the current request; that is, it can ensure that there is at least one true positive match for the request at the current node. In contrast to exact match forwarding, the results in BF-based forwarding may contain both true positive matches and false positive matches. If there is a false positive match, the router itself cannot know from which port the packet is correctly forwarded, and it may forward to the wrong hop if it randomly selects a positive match. We assume that there are i true positive matches, the remaining $(R - i)$ BFs will probably have false positives, and the FPR for each BF is ε. When there is a first filter claiming to have found the name, the probability that it is actually a false positive is noted as P, as calculated in Equation (6). In addition, this will result in a forwarding error. P will increase as R increases, and when the node maintains a lot of BFs, P will not be ignored, and the packet will most likely be forwarded incorrectly. Once the packet reaches the next routing node, if the correct match is not available (i.e., $i = 0$), then the success of the name being looked up here must be a false positive. This can be calculated according to Equation (6), $P = 1$. At this point, the packet has been incorrectly forwarded for two hops, and it is almost impossible for it to return to the correct forwarding path, as subsequent routers will not even be able to determine where the error started:

$$P = \frac{\varepsilon(R-i)}{\varepsilon(R-i)+i} = 1 - \frac{i}{\varepsilon(R-i)+i} \tag{6}$$

This incorrect delivery eventually leads to a delay in content acquisition. Usually, the node receiving the incorrectly delivered request is unable to find a subsequent forwarding address and will drop the request. The requester must rely on a timeout for retransmission,

but using the same name may still lead to an erroneous result, as the same false positives still occur on the BFs. Considering the high cost of incorrect deliveries, BF-based routing schemes are usually optimized in the following ways:

- Keep the false positive rate low at the filter level;
- Avoid relying only on filters for continuous multi-hop forwarding decisions;
- Design a reasonable multiple match resolution (MMR) strategy.

MMR strategies [14] can fundamentally prevent erroneous forwarding. In existing MMR strategies, the all-matched strategy has too much communication overhead, and the retransmission strategy requires an alternative addressing method. Ultimately, these strategies are designed for BF-based routing schemes, but they are limited by the bloom filter. Therefore, our approach selected the cuckoo filter to design a new error correction feedback strategy.

3.2. System Model

The existing Internet is organized by multiple autonomous systems (ASes), where the different ASes communicate through inter-domain routing, with the border routers of each AS interconnecting and collaborating on the forwarding of packets. Routing within the autonomous system is known as intra-domain routing, where the packets are forwarded only between different network devices in an AS. In contrast, our resolution system is also divided into an inter-domain resolution and intra-domain resolution. We also used these border routers as resolution nodes, because we believe that the IP-based Internet infrastructure will not be abandoned, and they have high stability in a dynamic network environment. To further reflect the topology of the underlying physical network, an AS can be divided into several control domains, each of which has an SDN controller responsible for directing the forwarding of requests. Thus, our scheme takes the controller as the basic intra-domain resolution node, with an advanced node within the AS for resolution between the control domains. A resolution system with a hierarchical network structure is shown in Figure 3.

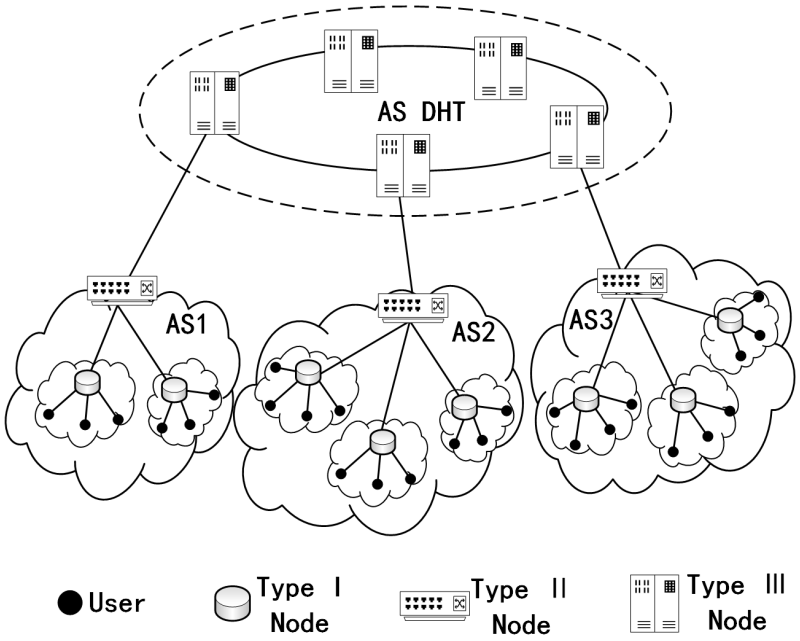

Figure 3. System model.

To allow IP services to coexist with ICN services, we defined a name resolution layer on top of the IP layer. Each resolution node runs both the IP routing protocols and the name resolution protocols, but different layers of nodes have their unique resolution functions. As the most basic resolution node, The Type I Node stores the mapping between the names

and addresses of content in the control domain. It will complete the name to address resolution process and direct the request to be forwarded to the closest available copy of the content. The Type II Node stores the correspondence between the name and the control domain in an AS, and it forwards the request to the Type I Node of the control domain, where the content is located if the requested content exists in the AS. The function of the Type III Node is to find the AS where the content is located. Therefore, all the border routers are required to form a DHT overlay network using the information provided by BGP, which can not only realize the inter-domain name resolution and routing functions but also ensure the scalability of the resolution system.

A content producer or requester can connect to a Type I Node as a user. Different from the IP services, the user must carry both the IP address and the content name when sending the ICN content request. The content name is a persistent and unique identifier for the name resolution layer. It is not allowed to be changed by the intermediate routing nodes during transmission. Moreover, our scheme supports both flat names and hierarchical names. The IP address is only used temporarily to identify location information, and the IP field in the packet will constantly be modified during routing and forwarding. The resolution node will take the content name for the resolution and direct the request to a better location according to this result. The destination address field will be modified to the address of the most recently available content, noted as *DestAddr*, or the address of the next resolution node that is more likely to know the location of the requested content. Algorithm 1 illustrates the name resolution and routing process.

Algorithm 1 Name resolution and routing.

```
1:  while True do
2:      if Type I Node then
3:          if Get the DestAddr then
4:              Forward(Packet, DestAddr)
5:              return
6:          else
7:              Forward(Packet, Type II Node)
8:          end if
9:      end if
10:     if Type II Node then
11:         if Get the Type I Node then
12:             Forward(Packet, Type I Node)
13:         else
14:             Forward(Packet, Type III Node)
15:         end if
16:     end if
17:     if Type III Node then
18:         Get the next Type II Node via DHT
19:         Forward(Packet, Type II Node)
20:     end if
21: end while
```

In our resolution system, the intra-domain resolution is populated to reduce unnecessary DHT routing. It takes advantage of the underlying physical network topology and IP routing protocols to forward requests to the nearest content source. Therefore, it is more efficient than conventional hierarchical DHTs, and this simple network structure also simplifies network management.

3.3. Simplified Adaptive Cuckoo Filter for a Resolution System

It is essential to introduce cuckoo filters to our resolution system for the following three reasons.

First, each Type I Node stores the mapping relationship between the content names and addresses in the control domain. As the user's access point and the most basic resolution node, these names will be queried at high frequencies, but they are very small compared with the huge name set. In other words, the Type I Node will receive a large number of requests that it cannot successfully find by itself. The CF is efficient in terms of lookup and gives a guaranteed correct negative answer, which is just suitable for this query scenario.

Second, the mapping relationship between the names and Type I Nodes is maintained on the Type II Node, which takes up a large number of storage resources. Setting up a CF for each Type I Node to store names can significantly improve space efficiency, which is the main reason for introducing filters in ICN name resolution and routing. However, to avoid relying on filters for multiple hops in a row during routing, we only used CFs on these two types of nodes.

Third, with the help of CF, a new MMR strategy can be designed to reduce the cost while obtaining the convenience brought by the filter. This strategy is described in detail in the next subsection.

The adaptive cuckoo filter (ACF) [33] has been shown to be effective in reducing the number of false positives in practice. It will recalculate the fingerprint that caused the error using an additional hash each time a false positive is found, thus reducing the possibility of subsequent errors for the same query. In the network, the probability of popular content being requested is much higher than the average, with 20% of requests often taking up 80% of the traffic. Therefore, this optimization of the ACF is very friendly to the resolution of popular names.

A large number of alternative hash functions is used in the ACF for computing fingerprints. The principle is that all fingerprints in a bucket are computed by the same hash function, and it uses extra bits as identifiers to identify the type of fingerprint in each storage bucket. The extra space taken up by the flag bits means that the storage capacity of ACF smaller. And Both reading and writing of flag bits will degrade the performance of the ACF. In addition, the ACF requires multiple cuckoo hash tables, and a main table to store the complete set of original elements. We kept the main table but changed the elements stored in it and simplified the ACF. We call it the SACF. Figure 4 shows our SACF and its corresponding main table.

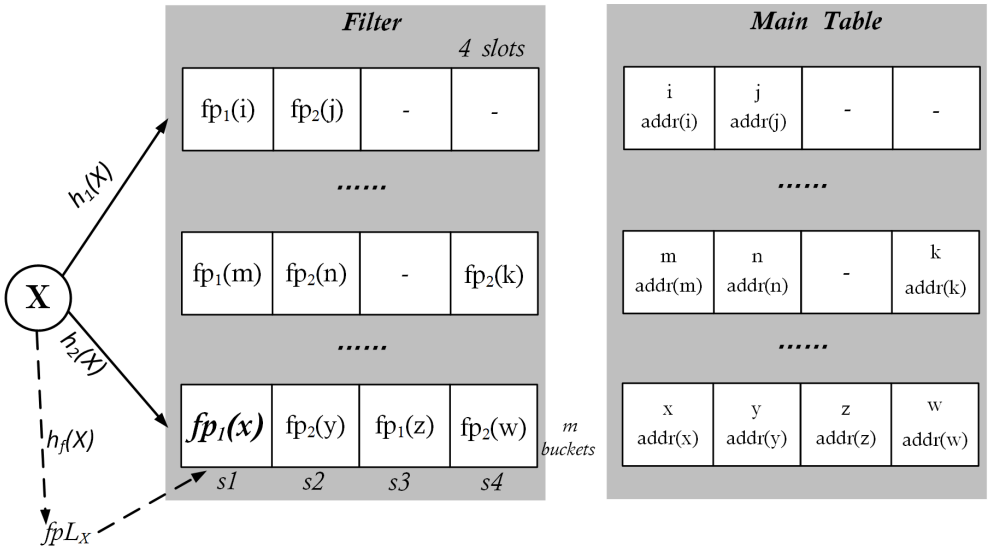

Figure 4. Correspondence between SACF fingerprints and main table elements.

The SACF is structurally identical to a normal CF. In order not to take up extra space by adding an identifier in the storage bucket, the SACF stores only two different types of fingerprints, with restrictions on where they can be located. Taking an element x as an example, its fingerprint may be either $fp_1(x)$ or $fp_2(x)$, both of a length f. In fact, we only need to compute a fingerprint fpL_x of a length $2f$, and the two short fingerprints can be obtained by separating the long fingerprint from the middle one. The relationship between the three fingerprints is shown in Equation (7), and this long fingerprint is also used to determine the positions of the candidate buckets as in Equation (8). However, the short fingerprints stored in the bucket will not be used directly during relocation, as it requires fetching the element from the main table and calculating the long fingerprint:

$$fpL_x = fp_1(x)2^f + fp_2(x) \tag{7}$$

$$\begin{cases} h_1(x) = hash(x) \\ h_2(x) = h_1(x) \oplus hash(fpL_x) \end{cases} \tag{8}$$

In addition, we made constraints on the storage positions of two types of fingerprints. The four slots in the bucket were recorded as s_1–s_4 for convenience, where $fp_1(x)$ can only be stored in s_1 and s_3, and s_2 and s_4 are used to store $fp_2(x)$. During the query, fpL_x is split into two short fingerprints $fp_1(x)$ and $fp_2(x)$, the same type of fingerprint is compared with the stored fingerprints, and a positive results is given when the match is successful. Algorithm 2 describes the SACF query process, which will also include the correction of false positives if error correction is enabled. This simplified design does not worsen the false positive rate, although intuitively, the two fingerprints are more likely to collide. The false positive rate of the SACF is recorded as FPR_{SACF}, and it includes collisions with two kinds of fingerprints that are independent of each other. When the loading rate is a, there is ac for each of the two fingerprints in the two candidate buckets, where c = 4 in our implementation. FPR_{SACF} is ultimately expressed as Equation (9):

$$FPR_{SACF} = (1 - (1 - \frac{1}{2^f})^{4a}) + (1 - (1 - \frac{1}{2^f})^{4a}) \approx \frac{8a}{2^f} \tag{9}$$

The two short fingerprints can be understood as different hash functions, which gives the SACF the ability to correct false positives by replacing the fingerprint representation. We assumed that x matched the fingerprint of element y and considered that $fp_1(x) = fp_1(y)$ without losing generality. A false positive is confirmed when the filter gives a positive result but the element x is not found in the main table. At this point, in the main table, the element y swaps places with its neighboring element z so that their fingerprints in the SACF are also recalculated. The fingerprints in the bucket where the false positive occurs in the SACF will change from $fp_1(y)$ and $fp_2(z)$ to $fp_1(z)$ and $fp_2(y)$, respectively, thus completing the replacement of the fingerprint type. After the error correction, the probability of a false positive ε_c is shown as Equation (10) when the same element x is queried consecutively, and a false positive will occur again only when $fp_1(x) = fp_1(z)$ or $fp_2(x) = fp_2(y)$ is satisfied. In practice, query requests are not completely ordered or random, and some popular content requests account for a large percentage. Error correction can reduce the false positives of these requests and will bring significant benefits. A more detailed proof can be found in [33]. To replace the short fingerprint, the ACF needs the main table that stores the complete elements and requires that the position of the element and the position of its fingerprint always coincide. For the SACF, in the main table, we store the correspondence between the content name and the address as an element:

$$\varepsilon_c = 1 - (1 - \frac{1}{2^f})^2 = \frac{1}{2^{f-1}} - \frac{1}{2^{2f}} \tag{10}$$

In fact, the SACF and the main table are the two structures we need to maintain on the Type I Node. When processing a name resolution request, the node first uses the SACF

to quickly locate the name in the main table and then performs another match to find the address where the name is located. If either of the two lookups fails, the request will be forwarded to the Type II Node. On a Type II Node, to save space, only a number of SACFs is stored instead of the main tables for intra-domain resolution. It is important to note that the SACF in a Type I Node stores the same names as the corresponding SACF in the Type II Node, and more strictly, they are identical filters.

Algorithm 2 Lookup for an item x.

```
ret = negative
Compute fpL_x Access buckets h1(x) and h2(x)
for j = 1 to 4 do
    m = (j − 1)%2 + 1
    n = 3 − m
    Compare fp in slot j with fp_m(x)
    if Match then
        ret = positive
        if Error correction enabled then
            if x is not in main table then
                Get item y in slot j from main table
                if m == 1 then
                    Get item z in slot k = j + 1 from main table
                end if
                if m == 2 then
                    Get item z in slot k = j − 1 from main table
                end if
                Swap y and z in main table
                Store fp_m(z) to slot j in filter
                Store fp_n(y) to slot k in filter
            end if
        end if
    end if
end for
return ret
```

3.4. SACF-Based Name Resolution and Routing

With the introduction of filters, our resolution system will also suffer from false positives. Therefore, we designed a new MMR strategy to ensure correct name routing. In addition, we added a support mechanism for content movement and deletion.

3.4.1. Error Correction Feedback Mechanism

In the whole system, the SACF is placed on only two levels of nodes for intra-domain resolution, and only the Type II Node maintains multiple filters. Such a design helps to limit where multiple matches occur. On a Type I Node, false positives do not even have an impact, as the main table will finalize the exact match of the name, thus fundamentally guaranteeing the correctness of the resolution result. Based on these characteristics, we designed an error correction feedback mechanism as the MMR strategy in our system.

During the name routing process, the error correction feedback mechanism comes into play when a request deviates from the correct path due to incorrect forwarding by the resolution node, and it helps to redirect the request back to the correct path through a process including the following stages:

- Sensing error forwarding: Although we already know that the error forwarding is due to multiple matches in the Type II Node, it does not know that an SACF has a false positive when it selects the next resolution node for forwarding. When the request is forwarded to a Type I Node, the SACF will also have a false positive because it stores the same fingerprint as the SACF with the false positive in the Type II Node. However,

the main table will return a lookup failure. At this point, the Type I Node perceives that this is a request forwarded incorrectly.

- Error correction and feedback: The Type I Node first corrects its SACF. It replaces the fingerprint that was incorrectly matched with another type of fingerprint, and the modified bucket of the SACF is repackaged with the request and sent back to the Type II Node.
- Synchronize and re-query: The Type II Node receives the repacking request, replaces the corresponding bucket in the SACF where the false positive occurred with the bucket carried in the repacking request and then continues with this name resolution and forwarding in the remaining SACFs.

Figure 5 shows the forwarding path of requests in case of false positives. This mechanism solves the pain point of the filter-based routing scheme through the one-to-one correspondence between elements and fingerprints in the SACF at the cost of a small communication overhead.

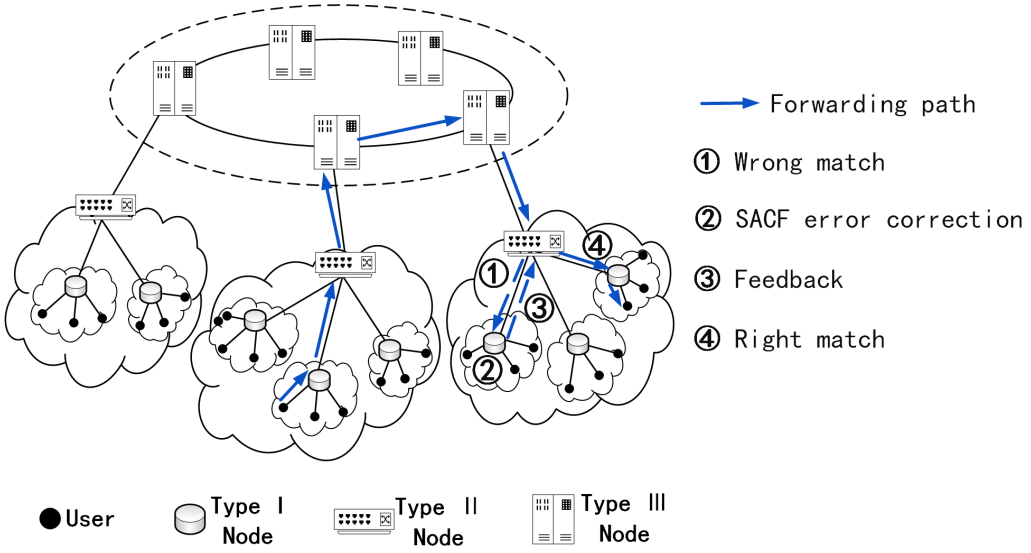

Figure 5. Error correction feedback mechanism.

3.4.2. Content Movement Support Mechanism

Although the SACF is a structure that naturally supports deletion, fingerprint collision may occur in deletion as in searching, resulting in false negatives eventually introduced by erroneous deletion, which is unacceptable. During deletion, the SACF removes a random fingerprint from one of the two candidate buckets. We would also prefer not to see this randomness because we hope that the SACF on a Type I Node and the SACF on the Type II Node representing this Type I Node are always identical, which is a prerequisite for error correction and feedback mechanisms to work. Thus, we designed the content movement support mechanism.

The deletion request is initiated by the user, which first performs a lookup on the Type I Node it accesses. To avoid erroneous deletion, we did not remove the fingerprint of the element after the SACF query returned a positive result, as we could not be sure whether this was just a hash collision. After the main table deletes the complete mapping relationship, the fingerprint of the SACF is removed according to the location relationship. In order to make the same SACF on the Type II Node also delete this fingerprint without introducing randomness, the modified bucket is synchronized to the corresponding SACF in the Type II Node, which is equivalent to another deletion. Due to the advantage that the SACF can be split into multiple buckets, adding a modified bucket after the deletion request will only occupy dozens of extra bits. In the last step, the mapping relationship between this content name and the Type II Node is deleted in the DHT network of the Type III Nodes, and the

content deletion ends. The main table deletes the original record as a start, which ensures that the deletion is reliable.

4. Performance Evaluation

In this section, we first test the performance of the SACF, including its false positive rate and query efficiency. Then, we use the NS3 network simulation platform to verify the effectiveness of the proposed mechanisms and compare the differences in terms of additional bandwidth and forwarding latency between the SACF-based scheme and the BF-based scheme.

4.1. Performance of the SACF

In this part, in order to test the performance of the SACF, we also implemented the other three filters—the BF, the CF and the ACF—for comparison. We initialized the four filters with the optimal configuration of the false positive rate by allocating the same size of storage space and limiting their maximum storage capacities to a similar size. Specifically, for the SACF, CF and ACF, we set the fingerprint length to $f = 8$, and accordingly, the optimal number of hash functions used for the BF was $k = 6$. For the ACF, we allowed it to provide up to four hash representations for each fingerprint. Therefore, the ACF required two extra bits to represent the type of fingerprint in each slot.

Although all filters are very efficient in terms of lookup, there are still some differences. After constructing several filters, we used the same set of candidate elements for insertion until the filters were filled to predetermined load rates. The load factor was defined as the ratio of the number of elements currently inserted into the filter to its capacity. Then, we tested the exact time required to perform queries under different loads. First, we chose a randomly generated dataset for testing. This dataset contained 100 million elements that were not stored in the filter, so we used it to test the time required for 100 million negative queries. The results obtained after several random tests are shown in Figure 6. As the efficiency of the filter in positive queries is equally important in our system, we used elements previously inserted into the filter with the same number of 100 million tests, and the results are shown in Figure 7.

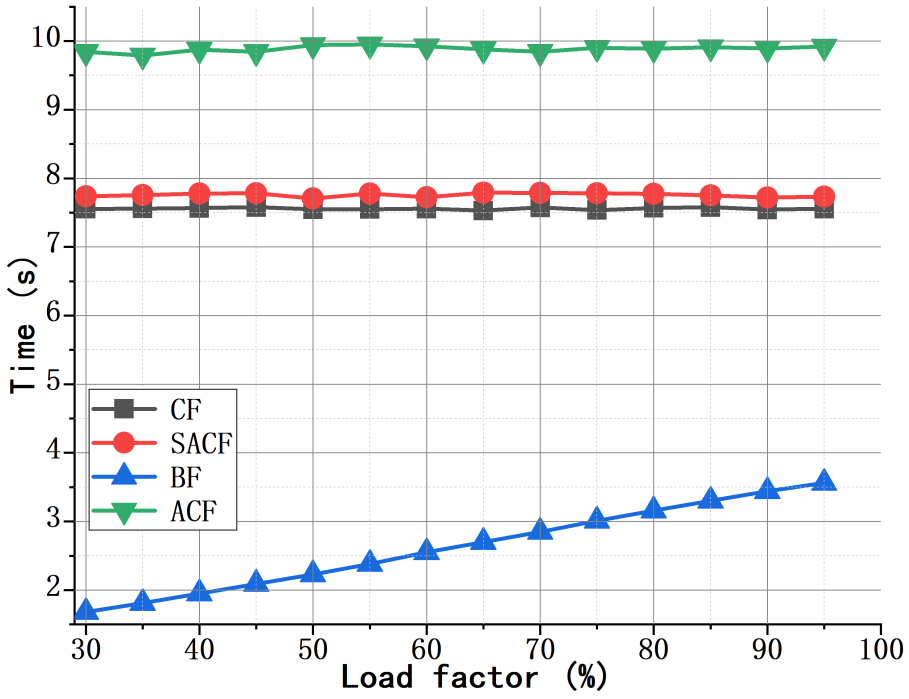

Figure 6. Times of 100 million negative queries at different load factors.

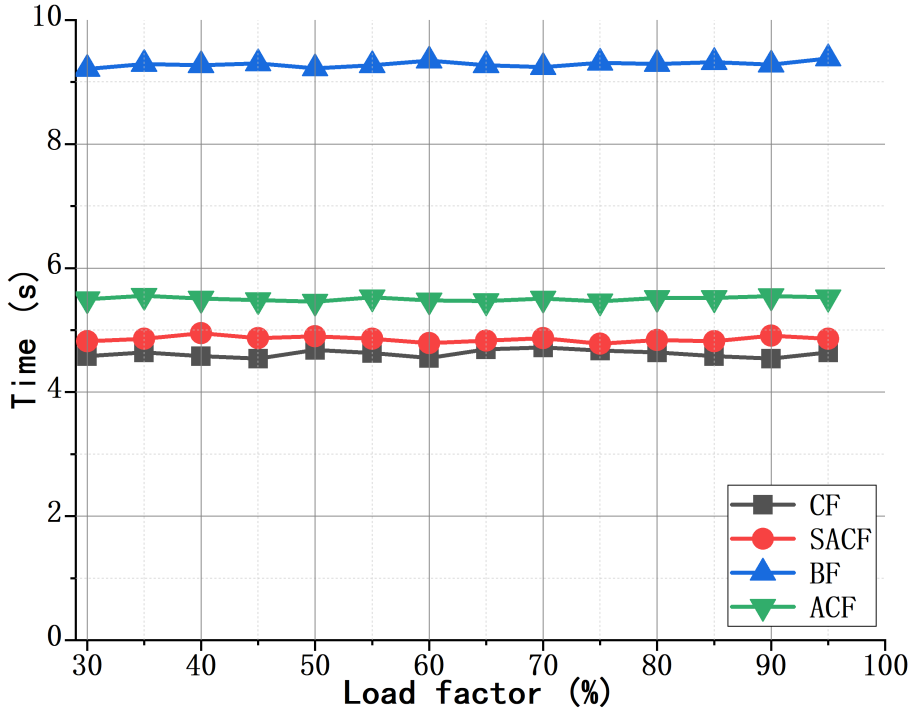

Figure 7. Times of 100 million positive queries at different load factors.

In a negative query, the BF required the least query time, since it only needed to find a "zero" among several bits representing the element being checked to return the result immediately, while it took longer to find the first "zero" bit as the number of bits set to "one" in the BF increased with the load, so its query time also increased. The CF, on the other hand, had a nearly horizontal query time curve because each negative query required a traversal to access a total of eight slots in two candidate buckets, which were not correlated with how many fingerprints were actually stored in the buckets. Thus, the CF's query time was not affected by the load. Similarly, the SACF and ACF also needed to traverse all slots of the two candidate buckets to complete a negative query. The SACF needs to calculate two types of fingerprints and use the same type of fingerprints in the slot for matching, which made the query take a little longer than the CF. The ACF needs to read the flag bits in the bucket at each query and then uses the corresponding hash function to calculate the fingerprint based on the flag bits. This process is more complicated than that of the SACF, so it consumes more time.

In a positive search, the query times of the four filters were almost unaffected by the load factor. Specifically, the BF always needs to calculate all k hash values to obtain the required query positions and verify whether these positions are "one". Therefore, this process consumed more time. Compared with the negative queries, the performance of the three CFs was much better. They only needed to find a fingerprint matching the element to be checked in the two candidate buckets and then return a success. However, their test times were also independent of the load condition, because empty slots cannot be skipped during the search. In the specific implementation, we used a fingerprint of "zero" in a slot to indicate that it was an empty slot. Although we did not enable the error correction function, the SACF still took some more time than the CF because it needed to calculate more hashes. Due to the flag bits, the ACF took over 20% more time to complete a search compared with the CF.

The false positive rate (FPR) represents the reliability of the filter. We used the WIDE2020 network dataset (http://mawi.wide.ad.jp/mawi, accessed on 01/01/2020) as the dataset to test the FPR of the lookup at different loads. This dataset contains many packets captured from a real-world network. We selected 10 million packets. The source IP and destination IP in a packet are regarded as a whole as one name. All names were divided

into two groups with no intersection. One group was used for insertion until the filters were filled to a predetermined load factor, and the other group was used for searching. In order to simulate the real situation in the network, the names being looked up were kept in the same order as they were in the original dataset. The statistical FPR results are shown in Figure 8.

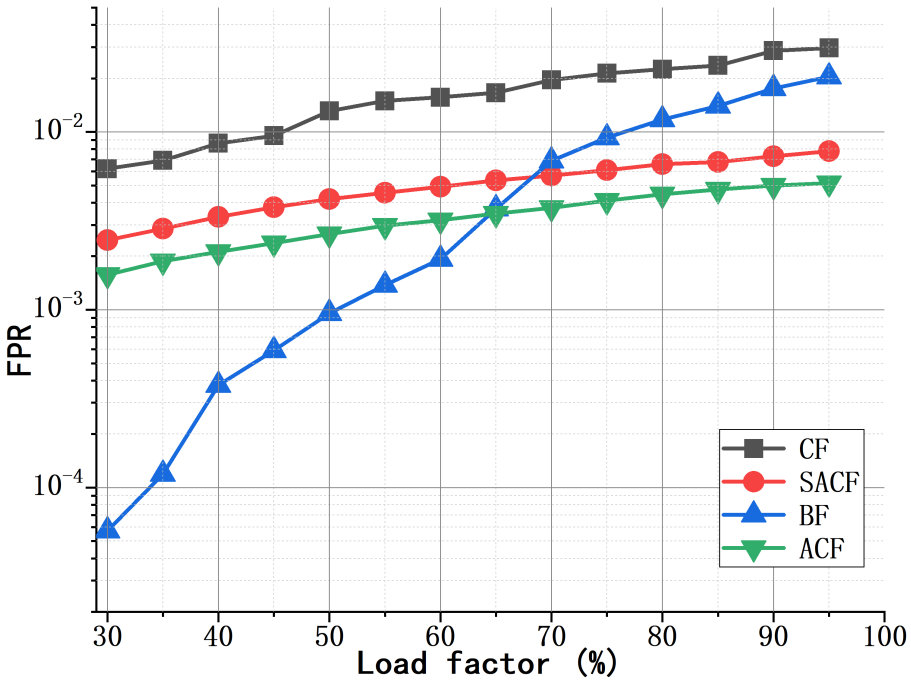

Figure 8. FPRs of filters at different load factors, where $f = 8$.

The FPRs for four filters decreased as the load decreased. The three curves for the filters varied almost linearly. The CF had the highest number of false positives in all cases. The BF's curve had a greater range of variation in the FPR. The BF was the optimal filter when the load factor was below 60% because the vacant bits of the BF contributed more in the true negative lookup. In practice, filters do not usually run at low loads because no one wants to leave a large amount of the available capacity empty. When the load exceeded 90%, the performance of the BF was closer to that of the CF, which made it less advantageous. Thanks to the error correction function, the SACF and ACF are well suited for use at high loads. As discussed before, once the type of fingerprint representation has changed, the probability that the same query is false again will be significantly reduced. As for the ACF, it had more selectable hash functions when correcting errors, so its performance was better than that of the SACF. Considering the space efficiency and query time of the SACF, its performance in terms of the FPR was excellent enough.

4.2. Performance of the SACF-Based Scheme

We implemented the SACF-based name resolution and routing scheme on the NS3 simulator. The test network included five ASes and several control domains in each AS. In the test network topology, a Type III Node, a Type II Node and R Type I Nodes were placed in each AS. All Type I Nodes were directly connected to the Type II Node, and a user was attached to each Type I Node. Since the Type II Node was directly connected to a Type III Node, we abstracted the function of the Type III Node for the Type II Node during implementation; that is, the Type II Node and Type III Node in each AS were actually the same node. At this point, the nodes between different ASes could not complete their communication. According to our system model, we used these five Type III Nodes to construct a small DHT network for inter-domain communication, where any two of them were only one hop away from each other. For the link configuration of NS3, the bandwidth

of all point-to-point direct links was set to 500 Mbps, and the communication delay was set to 1 ms. In these experiments, we only sent ICN name routing requests, so there was no impact of any other traffic and no network packet loss occurred.

In the SACF-based scheme, all SACFs were initialized with the same parameters, the capacity was set to 32768, and the fingerprint length was set to 8 bits. We still used the processed WIDE2020 dataset. We chose 30,000 different names as the insertion set. The names in the insertion set were registered to a randomly selected Type I Node, and the other Type I Nodes were registered with other names that did not belong to the insertion set. Finally, the load rate for all SACFs was over 90%. Then, each user sent 1000 resolution requests, and the names in the requests were obtained from the insert set with a certain probability. This probability is the frequency with which each data in the insertion set appears in the original dataset. Names with high frequencies in the dataset were more likely to be selected as names in the requests, which could simulate the impact of content popularity in the real network. As a comparison, we implemented the BF-based scheme [18] and used the same settings.

First, we adjusted the size of R to change the size of the network topology and tested the additional overhead. The additional bandwidth overhead was compared to the bandwidth consumption of the accurate name lookup. In the accurate name lookup, each request is correctly forwarded between resolution nodes until it reaches the target user. We used the product of the theoretical minimum number of forwards for a request $num(fw_{opt})$ and the size of the request packet $size(pkt)$ to represent the overhead of an accurate name lookup. In the SACF-based scheme, the actual number of forwards was higher, and the size of the error correction messages was larger than $size(pkt)$. The definition of the additional overhead is shown in Equation (11), where $num(fw)$ and $num(fw_c)$ represent the number of request packets forwarded and the number of error correction messages forwarded, respectively, and $size(pkt)$ and $size(pkt_c)$ represent the size of these two types of messages, respectively:

$$Additional \quad overhead = \sum_{pkt_{total}} \frac{num(fw) \times size(pkt) + num(fw_c) \times size(pkt_c)}{num(fw_{opt}) \times size(pkt)} - 1 \tag{11}$$

The proportion of additional overhead is shown in Figure 9. In the BF-based scheme, the additional overhead grew almost linearly as the network size increased. The BF-based scheme will forward messages to all nodes that give a positive result, which is actually a flooding approach. It is obvious that with the increase in the number of Type I Nodes, the scope of flooding will expand accordingly. The SACF-based scheme only requires forwarding to a truly correct node, although false positives also introduce additional message forwarding. The error correction feedback mechanisms can go some way to reducing the possibility of subsequent false positives, since false positives can be corrected after they are detected. This is very suitable for large-scale networks because its additional overhead fluctuates very little. At very small network sizes, the SACF-based scheme is more costly than the BF-based scheme. On the one hand, there are few false positives in small-scale networks. The SACF-based scheme solves the false positives by forwarding the request and recycling error correction messages, while the BF-based scheme only needs to copy a request for forwarding. This processing method makes the SACF-based scheme consume more bandwidth. On the other hand, the SACF-based scheme is not very effective despite its error correction. When the network size is small, there are few duplicate names in the requests, and there is no significant difference between the number of occurrences of popular names and cold names at this time. With the expansion of the network, another change is that the network contains more requests. The resolution process of frequently requested hot content will benefit from the previous error correction process.

Figure 10 compares the average latency of each resolution request in the two solutions. The BF-based scheme achieved excellent latency at the cost of a high overhead. As the number of nodes increased, although the BF-based scheme retrieved more filters, its latency

was almost independent of the network size because the filter lookup had only $O(1)$ for its complexity. In fact, this curve was almost the shortest path delay of the request from the sender to the receiver, because one of the replicated requests and the original request must have reached its destination by the optimal route. The latency in the SACF-based scheme was also not affected by the size of the network, which was actually for a different reason. As the network scaled up and the number of requests increased, the error correction mechanism controlled the number of false positives, so the vast majority of the requests was also not affected by false positives during forwarding. However, the average latency of the SACF-based scheme was still about 20% more than that of the BF-based scheme, which was the cost of the error correction feedback mechanism.

Figure 11 shows the communication overhead between nodes due to content deletion. This part only involved the information synchronization between a Type I Node and a Type II Node. For a Type I Node, we deleted a certain percentage of its stored content per unit of time. The BF-based scheme always synchronized the entire filter to the Type II Node, and as a baseline, its overhead was always kept at 100%, while the SACF-based scheme only sent the changed storage buckets to the Type II Node, which significantly reduced the overhead. When the content deletion rate was 1%, 5% and 25%, the overhead was 3.9%, 17.9% and 64.6%, respectively. Even in the case of random deletion, the ratio of the overhead to the deletion rate did not exceed four because a bucket could hold up to four names. Of course, the large-scale content deletion in this experiment is rare in practice. However, even if 10% of the content on a node is deleted per second, our scheme will reduce bandwidth consumption by 67% compared with the BF-based scheme.

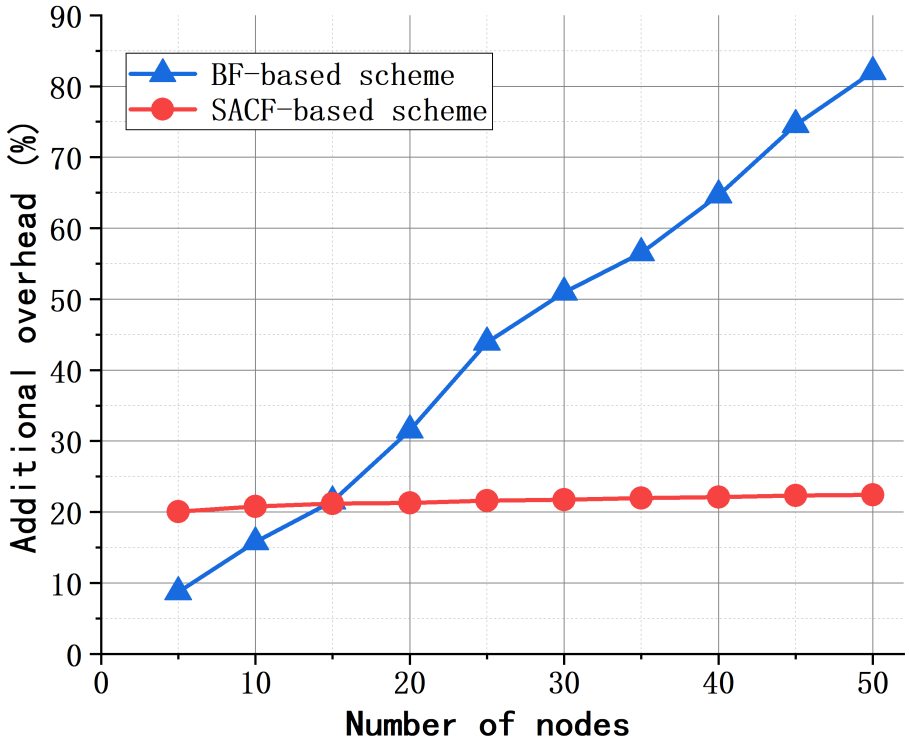

Figure 9. Additional forwarding overhead under varied network sizes.

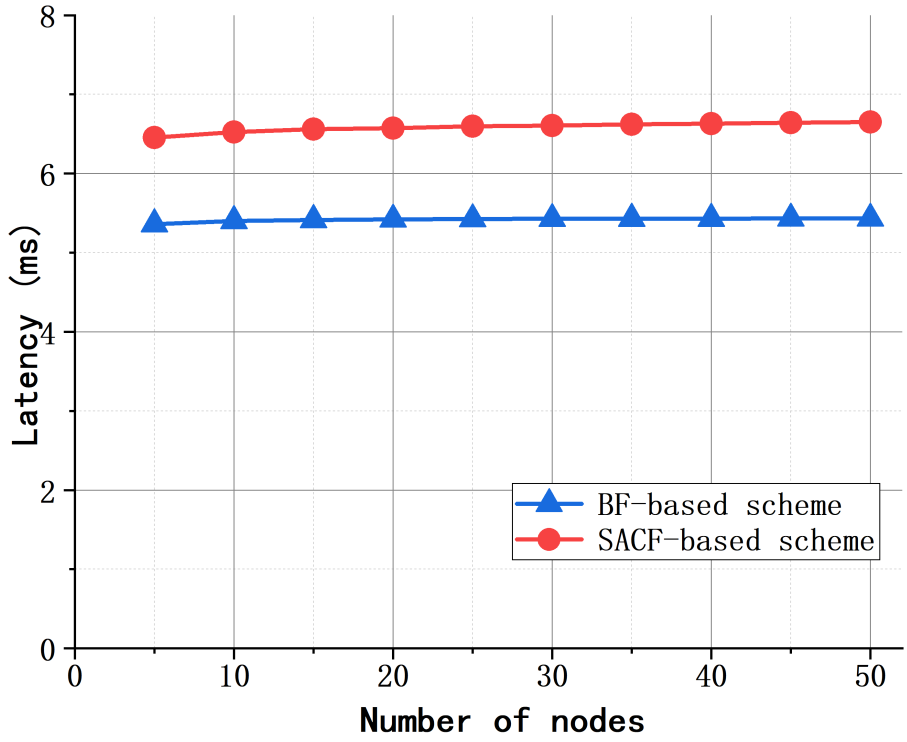

Figure 10. Forwarding latency under varied network sizes.

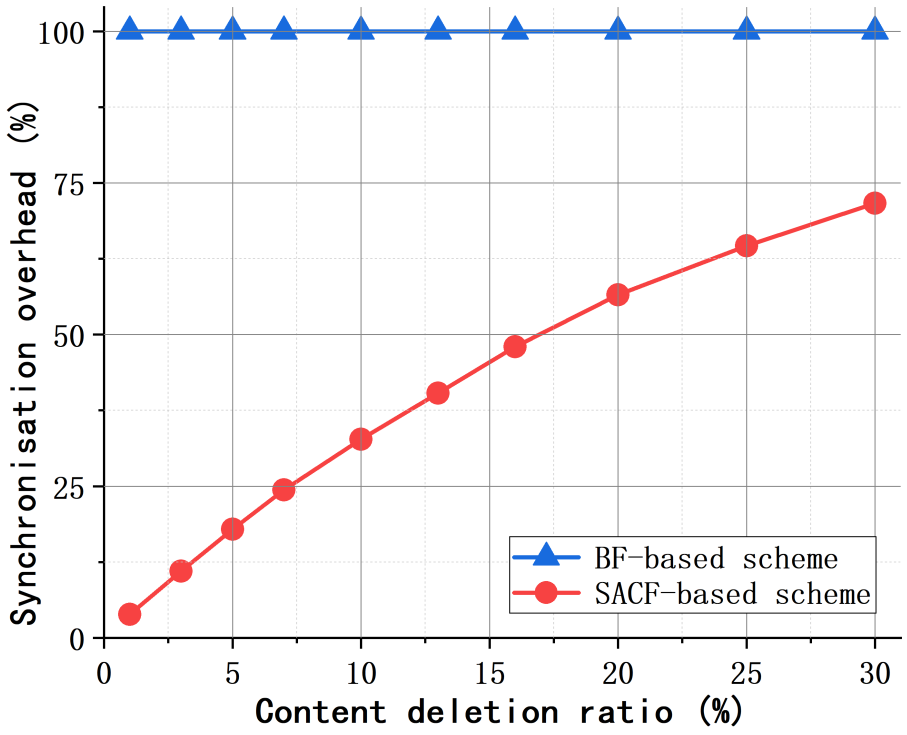

Figure 11. Synchronization overhead under varied deletion ratios.

5. Conclusions

This paper proposes a CF-based routing method to improve the efficiency of name resolution and routing in ICN and solve the problems of false positives and content mobility support in BF-based routing. Our method designs a hierarchical network architecture. Based on a traditional DHT scheme, we used network infrastructures to sense the physical topology and enable proximity forwarding of requests, which ensured scalability while

reducing the forwarding latency. To further accelerate forwarding and improve the memory efficiency of the resolution nodes, we proposed SACF, a CF variant that enables optimization of the false positive rate. In addition, we designed the error correction mechanism and content movement support mechanism between nodes to solve the impact of SACF false positives and effectively reduce the overhead in name routing.

Author Contributions: Conceptualization, W.L., J.Y. and Y.L.; methodology, W.L. and J.Y.; software, W.L.; writing—original draft preparation, W.L.; writing—review and editing, Y.L. and J.Y.; supervision, J.Y. and J.W.; project administration, J.Y. and Y.L.; funding acquisition, J.Y. and J.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by (1) the National Key R&D Program of China: Polymorphic Intelligent Network Environment (PINE) for Testing and Demonstrations (Project No. 2020YFB1806402) and (2) the Strategic Leadership Project of the Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We would like to express our gratitude to Yanxia Li and Zhaolin Ma for their support of this work.

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

References

1. Zhang, L.; Afanasyev, A.; Burke, J.; Jacobson, V.; Claffy, K.; Crowley, P.; Papadopoulos, C.; Wang, L.; Zhang, B. Named data networking. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 66–73. [CrossRef]
2. Venkataramani, A.; Kurose, J.F.; Raychaudhuri, D.; Nagaraja, K.; Mao, M.; Banerjee, S. Mobilityfirst: A mobility-centric and trustworthy internet architecture. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 74–80. [CrossRef]
3. Wang, J.; Chen, G.; You, J.; Sun, P. Seanet: Architecture and technologies of an on-site, elastic, autonomous network. *J. Netw. New Media* **2020**, *6*, 1–8.
4. 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.
5. Fotiou, N.; Nikander, P.; Trossen, D.; Polyzos, G.C. Developing information networking further: From PSIRP to PURSUIT. In Proceedings of the International Conference on Broadband Communications, Networks and Systems, Malaga, Spain, 4–6 November 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1–13.
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. Liu, H.; Azhandeh, K.; De Foy, X.; Gazda, R. A comparative study of name resolution and routing mechanisms in information-centric networks. *Digit. Commun. Netw.* **2019**, *5*, 69–75. [CrossRef]
8. Hoque, A.M.; Amin, S.O.; Alyyan, A.; Zhang, B.; Zhang, L.; Wang, L. NLSR: Named-data link state routing protocol. In Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking, Hong Kong, China, 12 August 2013; pp. 15–20.
9. Tortelli, M.; Grieco, L.A.; Boggia, G.; Pentikousisy, K. Cobra: Lean intra-domain routing in ndn. In Proceedings of the 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2014; pp. 839–844.
10. D'Ambrosio, M.; Dannewitz, C.; Karl, H.; Vercellone, V. MDHT: A hierarchical name resolution service for information-centric networks. In Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking, Toronto, ON, Canada, 15–19 August 2011; pp. 7–12.
11. Liu, H.; De Foy, X.; Zhang, D. A multi-level DHT routing framework with aggregation. In Proceedings of the Second Edition of the ICN Workshop on Information-Centric Networking, Helsinki, Finland, 17 August 2012; pp. 43–48.
12. Bloom, B.H. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **1970**, *13*, 422–426. [CrossRef]
13. Yu, M.; Fabrikant, A.; Rexford, J. BUFFALO: Bloom filter forwarding architecture for large organizations. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, Rome, Italy, 1–4 December 2009; pp. 313–324.
14. Rodrigues, A.; Steenkiste, P.; Aguiar, A. Analysis and improvement of name-based packet forwarding over flat id network architectures. In Proceedings of the 5th ACM Conference on Information-Centric Networking, Boston, MA, USA, 21–23 September 2018; pp. 148–158.

15. Papalini, M.; Carzaniga, A.; Khazaei, K.; Wolf, A.L. Scalable routing for tag-based information-centric networking. In Proceedings of the 1st ACM Conference on Information-Centric Networking, Paris France, 24–26 September 2014; pp. 17–26.
16. Papalini, M.; Khazaei, K.; Carzaniga, A.; Rogora, D. High throughput forwarding for ICN with descriptors and locators. In Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, Santa Clara, CA, USA, 17–18 March 2016; pp. 43–54.
17. Marandi, A.; Braun, T.; Salamatian, K.; Thomos, N. A comparative analysis of bloom filter-based routing protocols for information-centric networks. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 00255–00261.
18. Marandi, A.; Braun, T.; Salamatian, K.; Thomos, N. BFR: A bloom filter-based routing approach for information-centric networks. In Proceedings of the 2017 IFIP Networking Conference (IFIP Networking) and Workshops, Stockholm, Sweden, 12–16 June 2017; pp. 1–9.
19. Marandi, A.; Hofer, V.; Gasparyan, M.; Braun, T.; Thomos, N. Bloom filter-based routing for dominating set-based service-centric networks. In Proceedings of the NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–9.
20. Marandi, A.; Braun, T.; Salamatian, K.; Thomos, N. Pull-based bloom filter-based routing for information-centric networks. In Proceedings of the 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019; pp. 1–6.
21. Marandi, A.; Braun, T.; Salamatian, K.; Thomos, N. Network coding-based content retrieval based on bloom filter-based content discovery for ICN. In Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–7.
22. Lee, J.; Byun, H.; Lim, H. Dual-load Bloom filter: Application for name lookup. *Comput. Commun.* **2020**, *151*, 1–9. [CrossRef]
23. Broder, A.; Mitzenmacher, M. Network applications of bloom filters: A survey. *Internet Math.* **2004**, *1*, 485–509. [CrossRef]
24. Lim, H.; Lee, J.; Byun, H.; Yim, C. Ternary bloom filter replacing counting bloom filter. *IEEE Commun. Lett.* **2016**, *21*, 278–281. [CrossRef]
25. Komatsu, K.; Asaka, T. Routing information management for content oriented networks using Bloom Filters. In Proceedings of the 2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS), Hiroshima, Japan, 25–27 September 2013; pp. 1–5.
26. Patgiri, R.; Nayak, S.; Borgohain, S.K. Hunting the pertinency of bloom filter in computer networking and beyond: A survey. *J. Comput. Netw. Commun.* **2019**, *2019*, 1–10. [CrossRef] [PubMed]
27. Chen, H.; Jin, H.; Chen, L.; Liu, Y.; Ni, L.M. Optimizing bloom filter settings in peer-to-peer multikeyword searching. *IEEE Trans. Knowl. Data Eng.* **2011**, *24*, 692–706. [CrossRef]
28. Ariyoshi, T.; Fujita, S. Efficient processing of conjunctive queries in p2p dhts using bloom filter. In Proceedings of the International Symposium on Parallel and Distributed Processing with Applications, Taipei, China, 6–9 September 2010; pp. 458–464.
29. Gao, W.; Nguyen, J.; Wu, Y.; Hatcher, W.G.; Yu, W. Routing in Large-scale Dynamic Networks: A Bloom Filter-based Dual-layer Scheme. *ACM Trans. Internet Technol. (TOIT)* **2020**, *20*, 1–24. [CrossRef]
30. Katsaros, K.V.; Chai, W.K.; Pavlou, G. Bloom filter based inter-domain name resolution: A feasibility study. In Proceedings of the 2nd ACM Conference on Information-Centric Networking, San Francisco, CA, USA, 30 September–2 October 2015; pp. 39–48.
31. Fan, B.; Andersen, D.G.; Kaminsky, M.; Mitzenmacher, M.D. Cuckoo filter: Practically better than bloom. In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, Sydney, Australia, 2–5 December 2014; pp. 75–88.
32. Reviriego, P.; Martínez, J.; Larrabeiti, D.; Pontarelli, S. Cuckoo filters and Bloom filters: Comparison and application to packet classification. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 2690–2701. [CrossRef]
33. Mitzenmacher, M.; Pontarelli, S.; Reviriego, P. Adaptive Cuckoo Filters. *ACM J. Exp. Algorithm.* **2020**, *25*, 1–20. [CrossRef]