Efficient Lookup Schemes Based on Splitting Name for NDN

Qingtao Wu, Jinrong Yan, Mingchuan Zhang, Junlong Zhu, Ruijuan Zheng

College of Information Engineering, Henan University of Science and Technology, China
wqt8921@haust.edu.cn, yan_jr@163.com, zhang_mch@haust.edu.cn, jlzhu@bupt.edu.cn, rjwo@163.com

Abstract

Named Data Networking (NDN) is a novel networking architecture which retrieves the content by using its variable-length names. Name-based forwarding is a typical feature of NDN. Therefore, it has necessitated the design of fast forwarding lookup algorithms. In this paper, we propose an efficient name lookup scheme called *SNBS* (*S*plit the *N*ame into *Basis* and *Suffix*). In this scheme, we decompose *Basis* into many components. Each component of *Basis* is stored in a *C*ounting *B*loom *F*ilter (*CBF*). In addition, we also introduce a correlation verification method to ensure inherent correlation of all components of *Basis*. Furthermore, *Suffix* is processed by tree bitmap. By simulation, we show that our proposed lookup scheme can improve the lookup rate. Moreover, our scheme also reduces the bound of false positive probability.

Keywords: NDN, Name lookup, Counting bloom filter, Tree bitmap

1 Introduction

Named Data Networking (NDN) [1-2] is a future Internet paradigm which uses the unique content name to obtain content irrespective of IP addresses. It presents many issues in the traditional network, such as scalability, mobility and safety [3]. However, NDN directly uses the content as basic object of network processing, which brings with potential benefits [4]. Meanwhile, it has technical challenges. As interest packets forward by lookup the *L*ongest *P*refix *M*atch (*LPM*) of content names, the efficient and scalable lookup is an important problem in those challenges.

In NDN, implementing a name-based lookup scheme faces significant challenges [5]. A lookup in NDN is essential to scan dozens or hundreds of characters till finding the *LPM*. Lookup time is directly related to the length of name, which has challenged to the line-speed name lookup. That is more complicated than IP address matching. Moreover, *FIB* may become huge than IP forwarding tables so that the algorithm of name lookup may take intolerable time. Therefore, traditional prefix matching algorithms will be less efficient in content name lookup. Finally, the update of name prefix table in NDN more frequent than routing tables of traditional networks.

In facing these challenges, an efficient and scalable data structure is essential to implement the *FIB*. An effective data structure should have the charm of fast lookup, scalable update time and robust forwarding correctness. Some *CBF*-based schemes for names can be adopted as an effective solution [5]. However, existing methods based on the *CBF* incur false positive problems which can cause forwarding error. A tree-based scheme has the high flexibility, which is suit to the *suffix* [6]. To handle these issues, we design a data structure called *SNBS*. Based on the observation of name prefixes, these have two general characteristics: (1) the probability that a name contains a number of components, which is very low. A name length is relatively uniform; (2) a name prefix is also same in the different names, hence the redundant information is stored. Aiming at the characteristics above, we split the name into two stages that are similar to [7]. *SNBS* employs a *CBF* and a tree bitmap to achieve fast lookup and enhance forwarding correctness. Moreover, we introduce a correlation verification method to ensure inherent correlation of all components of *Basis*. The correlation verification reduces the bound of false positive problem brought by *CBF*. In summary, we make the following contributions:

(1) In this paper, we decompose *Basis* into many components that are stored in corresponding *CBF*. According to the characteristics of variable length, we store *Suffix* by using tree bitmap.

(2) *SNBS* has high accuracy of lookup by using the correlation verification method. The hash table can store the verification value. In addition, we can obtain forwarding interfaces and the position of *suffix* by verification in the hash table.

(3) We show the operations of inserting, lookup, and deleting items in proposed hybrid data structure.

(4) *SNBS* effectively reduces the bound of false positive problem in the process of the lookup. It is able to sustain relatively stable lookup performance with the increase of the number of name.

The rest of this paper is organized as follows. In Section 2, we present the related work. In Section 3,

*Corresponding Author: Jinrong Yan; E-mail: yan_jr@163.com
DOI: 10.3966/160792642019012001018

we present a fast longest prefix name lookup structure. In Section 4, we analyze the bound of false positive problem. In Section 5, we evaluate the proposed lookup structure. In Section 6, we conclude the paper.

2 Related Work

In recent years, there have been an increasingly large number of literatures on NDN [1-2]. However, in those studies, only a few are dedicated to name lookup in content

routers. In existed works, the software-based or hardware-based solutions are proposed to improve name lookup performance. In software, we survey related work in two aspects: (1) study on Bloom Filter to reduce the false positive, (2) study on tree data structure to reduce the depth of tree.

Bloom Filter is a space-efficient solution for fast *LPM*. Dharmapurikar et al. [8] first proposed a *LPM* algorithm that employs Bloom Filter to efficiently reduce the scope of lookup. A software engine based on hash tables was designed to achieve a fast forwarding lookup in [9]. Xiao et al. [10] proposed an approach stored multiple attributes of an item in parallel Bloom Filter. The same ideas are also used in our design. A parallel Bloom Filter is used in *URL* [11]. Nevertheless, the method based on Bloom Filter will cause relatively large false positive problem, which may forward the packet to the wrong interface.

Tree data structure is another effective way to achieve the *LPM*. A fast lookup framework was proposed and implemented for *LPM*, which uses extensible hybrid data structures [12]. Quan et al. [13] used the tree bitmap to degrade the depth of tree, which is useful to the variable length. However, the lookup scheme based on the tree will cause a high lookup cost because of the length of name.

Many hardware-based research works have been done since it has an advantage of hardware's parallelism. Lookup mechanisms that based on TCAM, which are presented in NDN [14]. However, it is more expensive form the cost point of view. In [15], a name component encoding method was proposed that implemented a GPU-accelerated lookup engine. The above research can greatly improve the process of lookup. However, they have compromised with the high cost and power consumption.

In this paper, we use the hybrid structure to reduce the bound of false positive. In addition, it can also improve the lookup speed of name. A name prefix is split into two stages, namely *Basis* and *Suffix*. In the first stage, the components among *Basis* are parallel lookup. Moreover, a correlation verification method is used to verify the correlation between components, which can reduce the upper bound of false positive problem. The length of *Basis* is decided by the split position *P*. In the second stage, we use tree bitmap to reduce the depth of tree. With this hybrid data structure and an optimized lookup algorithm, we can achieve efficient lookup performance as increasing the number of name prefixes.

3 Proposed Solution for Name Lookup

3.1 Split Model

Definition 1 (Split Model): Given the *FIB* and split position *P*, the *FIB* is split into two stages at the position *P*, namely, FIB_1 and FIB_2 ($FIB \xrightarrow{P} FIB_1 + FIB_2$).

Each input name is split into two parts at position *P*, namely, *Basis* and *Suffix* ($name \xrightarrow{P} Basis + Suffix$).

Name lookup is to seek for the forwarding interfaces of which the corresponding name prefix in *FIB* is determined by:

$$prefix = LPM(S_1 \bigcup S_2) = \begin{cases} LPM(S_1) & Suffix = \varnothing \\ t_1 \bigcup LPM(S_2) & Suffix \neq \varnothing \end{cases},$$

where

$$Basis \in S_1,\ Suffix \in S_2, S_1 = \{t \mid t \in FIB_1 \text{ and } t \le Basis\},$$
$$S_2 = \{t \mid t \in FIB, t \xrightarrow{P} t_1 + t_2, \text{and } t_1 = Basis, t_2 \le Suffix\}.$$

It is important to note that $t_i \le x$ shows the number of component of t_i is no more than x.

3.2 *SNBS* Structure

After the name split, it can effectively reduce the number of *Basis* for the polymerization of name. In this paper, we present an efficient name lookup scheme that combines *CBF* with tree bitmap. In the *SNBS*, the original *FIB* is split into FIB_1 and FIB_2 on the basis of Split Model at the position *P*. Each input name also is split at position *P* to product two shorter ones, namely, *Basis* and *Suffix*. They are looked up in FIB_1 and FIB_2 respectively. At last, their lookup result should be combined. In other words, the *Basis* of each name forms the FIB_1 and the *Suffix* of each name forms the FIB_2. To implement Split Model, there are three cases to be considered: (1) how to look up *Basis* in FIB_1; (2) how to look up *Suffix* in FIB_2; (3) How to combine these lookup results.

According to the definition of the Split Model, the lookup results of *Basis* exist in FIB_1. A *CBF* built on FIB_1 can be adopted to perform lookup of *Basis*. In addition, a tree bitmap built on FIB_2 is useful for the flexible lookup characteristics. *Basis* and *Suffix* are connected by the hash table. In addition to the above function, the hash table also shows forwarding interfaces and reduces the bound of false positive probability.

The hybrid NDN lookup structure is depicted in Figure 1. The storage of *Basis* is different from [8] in which the different lengths of name prefix are stored in

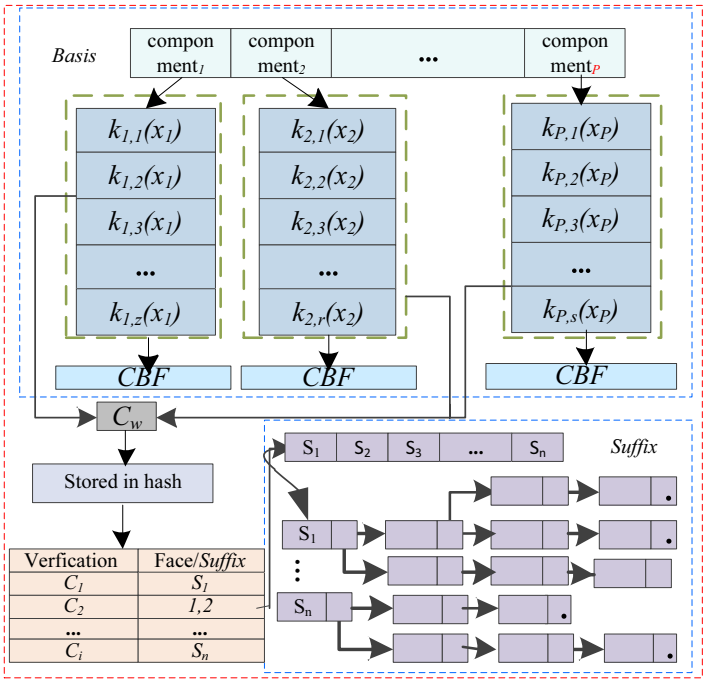

Figure 1. Hybrid name lookup structure

the different Bloom Filter. *SNBS* splits *Basis* into many components. Then each component is stored in a *CBF*. The *CBF* uses the segment-based form [10]. *SNBS* can perform the parallel lookup among components of *Basis*. *Suffix* uses tree bitmap for storage. If a name needs to be inserted or deleted, the corresponding *Basis* will be added or deleted from the *CBF*. In other words, the indexed positions of counters in *CBF* are incremented or decremented by 1. Similarly, the corresponding tree bitmap will be updated, which is specific detailed in [13, 16].

3.3 The Correlation Verification Mechanism

In the *Basis*, the parallel lookup of *CBF* is very effective. However, *Basis* is split into many components. To judge the relevance between components of *Basis*, we introduce the correlation verification mechanism in the *Basis*. The presence of a name prefix can be verified when it presents in both *CBF* and the hash table. The correlation verification is based on the weighted way. A method of generating the verification value c_i can distinguish verification values from different components in which the sequential hash functions are assigned different weights. In *Basis*, assuming that the number of parallel Bloom Filters is P. The size of the i-th *CBF* is m_i. The set of hash functions in the i-th *CBF* is $K_i = \{k_{i,1}, k_{i,2}, \cdots, k_{i,l}\}$, therefore, the i-th *CBF* contains $|K_i|$ independent hash functions. Thus the range of a hash function is $[1, m_i / |K_i|]$ in a *CBF*. It is important to note that $m_i \geq |K_i|$. The range of a hash function is depicted in Figure 2. A proportion of *Basis* in the total name is u. x_i is the i-th component of name x. The j-th hash value of x_i is computed by the j-th hash function in the *CBF*, which represents $k_{i,j}(x_i)$. The hash function is a random variable following the Uniform Distribution.

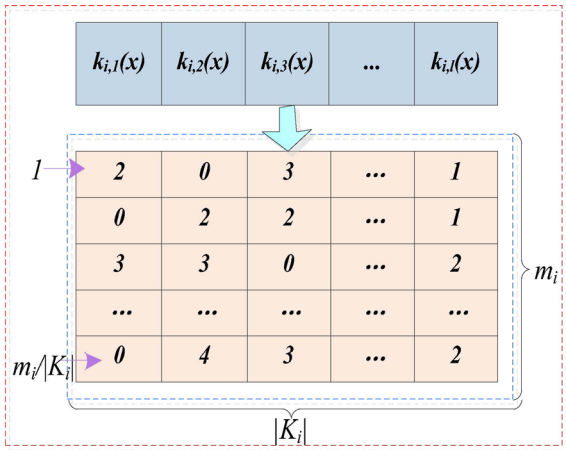

Figure 2. The range of a hash function for segment-based form

Verification value of the i-th component of *Basis* of x is

$$c_i = \sum_{j=1}^{|K_i|} 2^{-uj} k_{i,j}(x_i) \quad \textbf{(1)}$$

where

$$c_i \in \left[\frac{1}{2^u - 1}\left(1 - \frac{1}{2^{u|K_i|}}\right), \frac{m_i}{|K_i|(2^u - 1)}\left(1 - \frac{1}{2^{u|K_i|}}\right) \right]$$

The verification value of x is given by

$$C_w = \sum_{i=1}^{w} c_i \quad \textbf{(2)}$$

where

$$C_w \in \left[\frac{1}{2^u - 1} \sum_{i=1}^{w} \left(1 - \frac{1}{2^{u|K_i|}} \right), \sum_{i=1}^{w} \frac{m_i}{|K_i|(2^u - 1)} \left(1 - \frac{1}{2^{u|K_i|}} \right) \right]$$

In equation (2), w denotes the number of components of *Basis* for name x and $w \le P$. Because of the different size of the *CBF*, we have not amalgamate with the scope with c_i. Verification value will be inserted into the hash table. By the verification value, it can find the corresponding *Suffix* or the forwarding interfaces.

3.4 Name Lookup

Table 1 presents the looking items in the *SNBS* structure. There have two cases: (1) the length of name prefix is less than or equal to position *P*, only the *Basis* needs to be looked up, which is processed by the *CBF*; (2) the length of name prefix is greater than *P*, the lookup will be process in the two stages. A given name is divided into two parts by the function of *GetIsolation*, namely, x^B and x^S. For *Basis*, verification value of name C_w is initialized to zero. We first compute $|K_i|$ hash functions for the *i*-th component of x^B in parallel. We use array $Bv[i]$ that records effective validation values for a component. If the *j*-th counter has a value of zero, the length of longest prefix of the name is less than *i* as shown in lines 3-8. In this process, variable *y* records the minimum subscript of *CBF* that the counter has a value of zero. If not, the verification value C_w is computed. The elements of stack are stored in order from the small to the large. It obtains the forwarding faces by the stack as shown in lines 18-27. If a counter hasn't a value of zero until the value of *y* is greater than or equal to *P*, it looks up the corresponding *Suffix* by the verification value of *Basis* in the hash table, then it will return the forwarding interfaces as shown in lines 22-24. The verification value C_w is produced by iteration as shown in lines 3-10. In the algorithm 1, the component of *Basis* is checked twice from the *CBF* and hash table. It can effectively reduce the false positive.

Table 1. The algorithm for looking items

Procedure QueryItem(Input: Item x)	
1. $(x^B, x^S) \leftarrow GetIsolation(x)$	15. *put the* C_w *into the stack*
2. *for each component i of* x^B *do*	16. $C_w = C_w + B_v[i]$
3. *initialize* $C_w=0$, $y=P+1$	17. *end for*
4. *for*($j=1, j \le |K_i|, j$++) *do*	18. *if(stack is empty)*
5. $l=k_{i,j}(x_i)$	19. *Return the default face*
6. *if*($CBF[l]=0$ *and* $y>i$)	20. else
7. $y=i$	21. *if*($y \le P$)
8. *break;*	22. *take the top element of the stack* C_w
9. *end if*	23. *a hash check for the forwarding the face by the* C_w
10. $C_w = C_w + l/2^{uj}$	24. *Return the forwarding face*
11. *end for*	25. *else*
12. $B_v[i] = C_w$	26. *look up the suffix that corresponding the Basis*
13. $C_w = B_v[1]$	27. *Return the forwarding face*
14. *for*($i=1; i<y; i$++)	28. *end procedure*

Figure 3 shows the lookup process of *SNBS*. Name is divides into two parts for the lookup. In *Basis*, a component is associated with a *CBF* which can be parallel lookup with other *CBF*. Hash table stores the verification value to verify the correlation, look up the forwarding interfaces and ensure the position of *Suffix*. Hence it achieves the lookup for the hybrid structure.

3.5 Update Scheme

SNBS provides a relatively stable update process. The update operation in *SNBS* includes the insertion and deletion operation. The update process is triggered when a new entry needs to be inserted or an expired one needs to be deleted in the *FIB*. In this paper, the number of *Basis* will reduce for splitting, which makes the possibility of the update in *CBF* will be dramatically reduced. In the algorithm, the size of each *CBF* can be different, and the number of hash function

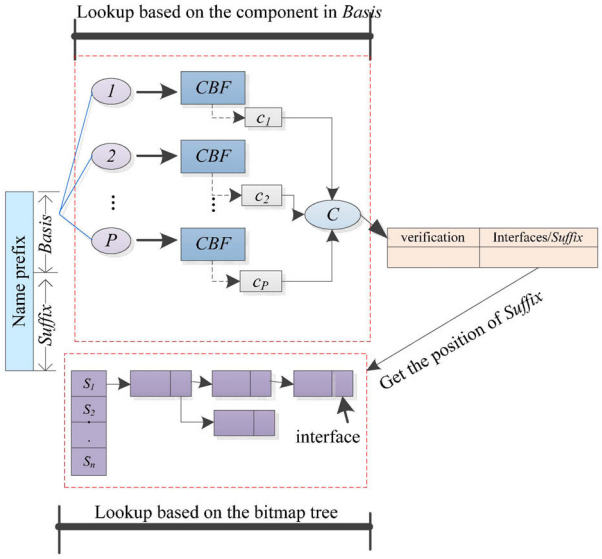

Figure 3. Match process

of a *CBF* also can be different from other *CBF*. Hence, we can modify the size of *CBF* according to the number of name prefix.

3.5.1 Inserting Items

The algorithm for inserting entries in *SNBS* is given in Table 2. When an item is inserted into the *SNBS*, there are following cases: (1) if the length of the longest prefix *t* is less than the split position *P*, it needs to update *CBF* first. The counters of *CBF* are incremented by 1, which can be done by computing hash value of a component. Then it generates the verification value based on weighted method, and finally inserts the verification value into the hash table, as shown in lines 3-7. Under this circumstance, if the x^S exists, the algorithm will continue to update tree bitmap. We will allocate the new root node, then we carry out an insertion in tree bitmap by setting the associated bits to one [13, 16], as shown in lines 8-10; (2) if the length of longest prefix t is more than the split position *P*, it just needs to update the tree bitmap and does not need to do anything to *CBF*. In other words, the *Basis* has already existed, as shown in lines 12-13.

Table 2. The algorithm for inserting entries

```
Procedure Insert_Item(Input: Item x)
1.  (x^B, x^S)←GetIsolation(x)
2.  t←Lookup(x^B)
3.  if(t <P and t !=the number of component in x^B)
4.      l=k_{i,j}(x_i)
5.      CBF[l]++
6.      C_w=C_w+l/2^{uj}
7.      insert C_w into the hash table
8.      if(x^S is not empty)
9.          locate the new allocated root node S
10.         update the bitmap by setting the associated bits
            to one
11. else
12.     locate the leaf node contains an insertion
13.     update the bitmap by setting the associated bits to one
14. end procedure
```

3.5.2 Deleting Items

The algorithm for deleting entries in *SNBS* is given in Table 3. When an item is deleted from the *SNBS*, there are following cases: (1) if the length of the longest prefix *t* is less than the split position *P*, the counters of *CBF* are decreased by 1, which can be done by computing hash value of component. The verification value will be deleted from the hash table, as shown in lines 3-7. In addition to, if the corresponding tree bitmap of verification value exists, we carry out a deletion in tree bitmap by setting the associated bits to zero in lines 8-9; (2) if the length of the longest prefix *t* is more than the split position *P*, we carry out a deletion in tree bitmap by setting the associated bits to zero [17]. Due to the aggregation of names, the operation of *CBF* needs to consider the condition for the common *Basis* of different *Suffix*. Accordingly, if the tree bitmap is empty after we delete the x^S from the tree, we carry out the deletion for *CBF*, as shown in lines 13-18.

Table 3. The algorithm for deleting entries

```
Procedure delete_Item (Input: Item x)
1.  (x^B, x^S)←GetIsolation(x)
2.  t←Lookup(x^B)
3.  if(t <P)
4.      l=k_{i,j}(x_i)
5.      CBF[l]--
6.      C_w=C_w+l/2^{uj}
7.      delete C_w from the hash table
8.      if(x^S is not empty)
9.          update the bitmap by setting the associated bits to zero
10. else
11.     update the bitmap by setting the associated bits to zero
12.     if(the tree bitmap is empty)
13.         l=k_{i,j}(x_i)
14.         CBF[l]--
15.         C_w=C_w+l/2^{uj}
16.         delete C_w from the hash table
17. end procedure
```

4 False Positive Analysis

In [8], the false positive of *CBF* gets the minimum of $(1/2)^{|K_i|}$ when $|K_i| = (m_i / n_i)\ln 2$. n_i stands for the number of name in the *i*-th *CBF*. Assume the false positive probability of the *i*-th *CBF* of *SNBS* is f_i. Obviously, f_i gets the minimum of $(1/2)^{|K_i|}$ when $|K_i| = (m_i / n_i)\ln 2$. The number of name prefix is *n*. F_{c_i} is the conflict probability of verification value of the name prefix which contains *i* components. Because of the introduction of a correlation validation mechanism, the false positive probability of *SNBS* for each name matching f_{CBF_i} is:

$$f_{CBF_i} = f_1 \cdot f_2 \cdots f_i \cdot F_{c_i} \quad \forall i \in \{1,2,3,\cdots,P\} \tag{3}$$

The expectation of c_i in a *CBF* is

$$E(c_i) = \frac{1}{2}\left(\frac{1+m_i/|K_i|}{2^u-1}\right)\left(1-\frac{1}{2^{u|K_i|}}\right) \tag{4}$$

The variance of c_i is

$$V(c_i) = \frac{1}{12}\left(1-\frac{1}{2^{u|K_i|}}\right)^2\left(\frac{m_i/|K_i|-1}{2^u-1}\right)^2 \tag{5}$$

Verification value C_i is the sum of *i* independent random variable. According to central limit theorem in [16], the verification value C_i in the *Basis* obeys the

normal distribution:

$$C_i \sim \mathcal{N}(\alpha, \beta) \tag{6}$$

where

$$\alpha = \sum_{j=1}^{i} \frac{1}{2}\left(\frac{1+m_j/|K_j|}{2^u-1}\right)\left(1-\frac{1}{2^{u|K_j|}}\right)$$

$$\beta = \sum_{j=1}^{i} \frac{1}{12}\left(1-\frac{1}{2^{u|K_j|}}\right)^2\left(\frac{m_j/|K_j|-1}{2^u-1}\right)^2$$

Comparing with [10], the upper bound of F_{c_i} in this paper is:

$$\begin{aligned} F(C_i) &= \sum_{i=1}^{n} C_i \le p\left(E(C_i)-\frac{n}{2} \le C_i \le E(C_i)+\frac{n}{2}\right) \\ &= p\left(-\frac{n}{2\delta} \le \frac{C_i - E(C_i)}{\delta} \le \frac{n}{2\delta}\right) \\ &= 2\Phi\left(\frac{n}{2\delta}\right)-1 \end{aligned} \tag{7}$$

$$F(C_i) \le 2\Phi\left(\frac{n}{2\delta}\right)-1$$

$$= 2\Phi\left(\frac{n}{2\sqrt{\sum_{j=1}^{i}\frac{1}{12}\left(1-\frac{1}{2^{u|K_j|}}\right)^2\left(\frac{m_j/|K_j|-1}{2^u-1}\right)^2}}\right)-1$$

where Φ represents cumulative distribution function of the standard normal distribution. δ denotes the standard deviation, namely $\delta^2 = \beta$.

For the longest prefix matching of *SNBS*, the bound of false positive probability f_{SNBS} is given by

$$f_{SNBS} \le 1-\prod_{j=1}^{P}(1-f_{CBF\,j}) \tag{8}$$

Furthermore, we have

$$\begin{aligned} f_{SNBS} &\le \sum_{j=1}^{P} f_{CBF\,j} \\ &= \sum_{j=1}^{P}(f_1 \cdot f_2 \cdots f_j \cdot F_{c_j}) \\ &= \sum_{j=1}^{P}\frac{1}{2^{|K_1|+|K_2|+\cdots+|K_j|}}F_{c_j} \end{aligned}$$

In addition, the false positive probability f_S in the [8] is given by

$$f_S \le \sum_{j=1}^{P}\frac{1}{2^{|K_j|}} \tag{9}$$

Comparing with [8], we can see that the bound of false positive probability of *SNBS* are lower by expressions (8) and (9) when the storage space and the number of set elements are simultaneous.

5 Performance Evaluation

In this section, we evaluate the performances of *SNBS* in terms of memory consumption, lookup time consumption and update performances. In addition, we choose Bloom-Hash (BH) [8], Hash Table (HT) [9] for comparison. The BH, HT and *SNBS* schemes are implemented in C++ language. The performances are measured in the PC with an Intel Core 2 Duo CPU of 2.8 GHz and DDR2 SDRAM of 4 GB. In the experiments, the number of name is 1M~10M, the number of *CBF* are 5. The number of hash functions is 6. According to [13], we let *P* be equal to 5, which is the optimal value. We utilize the domain name information from the Blacklist [18] and collect a larger number of *URLs* to build the experimental names dataset. For each name, we randomly associated with one forwarding face. The component number of names uniformly ranges from 2 to 7.

(1) Memory consumption

The memory consumption of the three methods on different prefix table sizes is illustrated in Figure 4. It observes that the memory consumption increases with the scale of name prefix. The BH is lower than other methods. The HT has the intermediate behaviors. Compared with other approaches, the memory consumption is defects of *SNBS*. However, *SNBS* only takes more 18.79% than BH on average with the scale of prefix. And it takes more 5.87% than HT on average with the scale of prefix. Note that the correlation verification mechanisms that uses hash table to maintain real verification values of *Basis* instead of its hashed results in a *CBF*. However, it doesn't require much memory cost since selecting the suitable position *P* can optimize the memory cost.

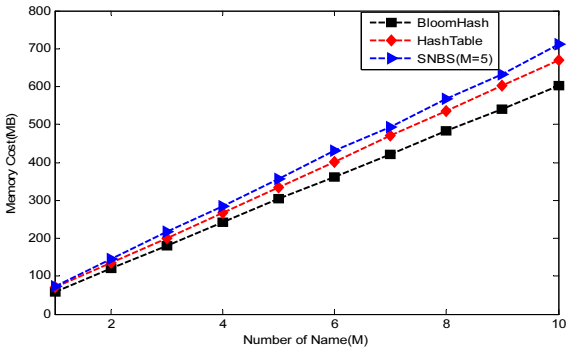

Figure 4. Memory cost

(2) Lookup time consumption

In the evaluation, we input 100K names at each time to get the total lookup time. Then we can get the average lookup time of name prefix by the total lookup

time. Figure 5 presents the comparison in terms of lookup time consumption. BH has more high time than HT and SNBS. The reason is that the BH and HT have conduct time consumption and need to cope with a number of conflicts. The component of *Basis* in *SNBS* can be parallel processing and it is split into shorter than whole name. In addition, the verification value based on weight can reduce the conflicts in some degree. The average lookup time of BH is 1.19us~1.27us, which can process the name 787k~840k per second. The HT's average lookup time is 1.04us~1.23us, which can process the items 813k~961k per second. Compared to the above two methods, *SNBS*'s average lookup time is 0.87us~1.07us, which can process the 935k~1149k per second. Although the memory consumption is relatively large, the speed is remarkably improved.

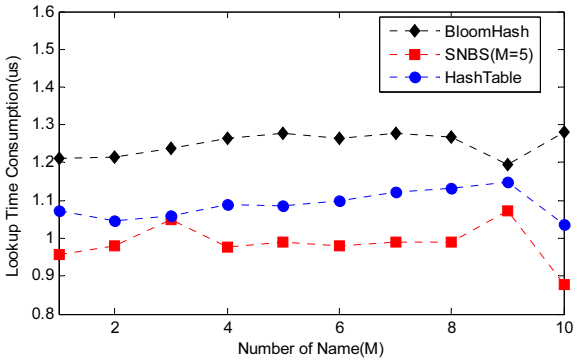

Figure 5. Lookup time consumption

(3) Update performance

The update process contains the insertion and deletion. The inserted time is calculated by inserting 100K new names to the BH, HT and *SNBS* for a different name scales. Figure 6 and Figure 7 show the update process. The average time of insertion for BH is 4.35 us~6.48 us, it can insert the name prefix 154k~230k per second. The average time of insertion for HT is 2.82us~6.03us, it can insert the name prefix 166k~355k per second. The average time of insertion for *SNBS* is 3.65us~4.94us, it can insert the name prefix 203k~274k per second. It can observe that *SNBS* can achieve a stable performance than other means.

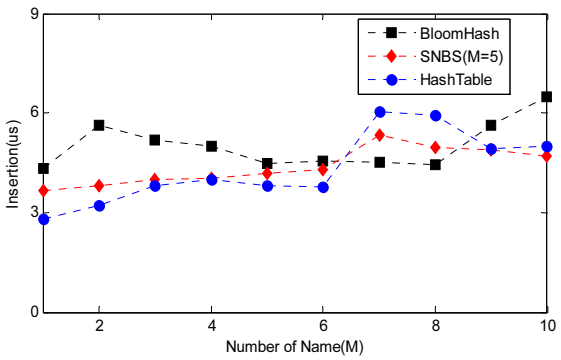

Figure 6. The update of insertion

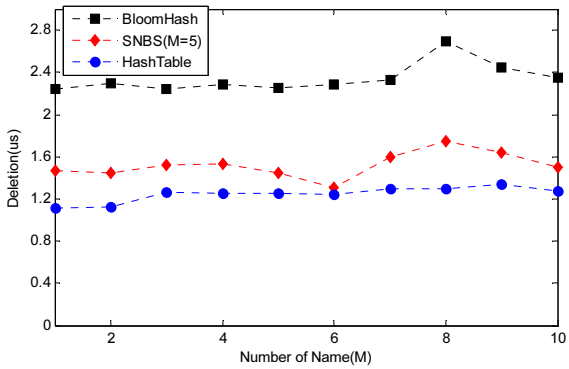

Figure 7. The update of deletion

The deleted time is calculated by deleting 100K names from the BH, HT and *SNBS* for a different name scales. From the Figure 7, the average time of deletion for BH is 2.24us~2.35us, it can delete the name prefix 425k~446k per second. The average time of deletion for HT is 1.11us~1.34us, it can delete the name prefix 746k~900k per second. The average time of deletion for *SNBN* is 1.31us~1.75us, it can delete the name prefix 571k~763k per second. The update of *SNBS* is better than the BH in general. The reason is that the split of prefix shortens wholes items which reduces the frequency of updates *CBF* and makes the processing more efficient. The correlation verification mechanism is introduced which may increase update time than HT. However, we can know correlation verification mechanisms that reduce the bound of false positive probability of *SNBS* from the section 4.

In summary, these experiments show that *SNBS* achieves relatively high processing speeds. Although there is variation, say generally, the processing speed is relatively steady.

6 Conclusion

This paper proposes a lookup scheme called *SNBS* to improve the lookup performance of *FIB* for packet forwarding in NDN. The *FIB* is split into FIB_1 and FIB_2. FIB_1 is represented by the *CBF* and FIB_2 is represented by the tree bitmap. Each input name is split at position *P*, namely *Basis* and *Suffix*. *Basis* is decomposed into many components to reduce false positive probability. In addition, we utilize the correlation verification mechanism to ensure inherent correlation of all components of *Basis*. Computing the hash of *Basis* can carry out in parallel to reduce the lookup time. By the analysis, we can find the *SNBS* achieves a lower bound of false positive. Evaluation results indicate that *SNBS* achieves a relatively high lookup speed.

Acknowledgments

This work is partially supported by the National

Natural Science Foundation of China (NSFC) under Grants No. U1404611, No. U1604155, and No. 61370221, in part by the Program for Science & Technology Innovation Talents in the University of Henan Province under Grants no. 16HASTIT035, and in part by Henan Science and Technology Innovation Project under Grant No. 164200510007 and No. 174100510010, and in part by Henan Province University-industry Project under Grant No. 172107000005.

References

[1] L. X. Zhang, A. Afanasyev, J. Burke, B. C. Zhang, Named Data Networking, *ACM Sigcomm Computer Communication Review*, Vol. 44, No. 3, pp. 66-73, July, 2014.

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman, A Survey of Information-centric Networking, *IEEE Communications Magazine*, Vol. 50, No. 7, pp. 26-36, July, 2012.

[3] R. J. Zheng, M. C. Zhang, Q. T. Wu, W. Y. Wei, C. L. Yang, A3srC: Autonomic Assessment Approach to IOT Security Risk Based on Multidimensional Normal Cloud, *Journal of Internet Technology*, Vol. 16, No. 7, pp. 1271-1282, January, 2012.

[4] S. Gao, H. K. Zhang, A. Sun, Y. M. Huang, Supporting Scalable Source Mobility Management for Named Data Networking, *Journal of Internet Technology*, Vol. 17, No. 4, pp. 619-632, January, 2016.

[5] H. C. Dai, J. Y. Lu, Y. Wang, T. Pan, B. Lin, BFAST: High-Speed and Memory-Efficient Approach for NDN Forwarding Engine, *IEEE/ACM Transactions on Networking*, Vol. 25, No. 6, pp. 1235-1248, April, 2017.

[6] W. Quan, C. Q. Xu, J. F. Guan, H. K. Zhang, L. A. Grieco, Scalable Name Lookup with Adaptive Prefix Bloom Filter for Named Data Networking, *IEEE Communications Letters*, Vol. 18, No. 1, pp. 102-105, January, 2014.

[7] Y. B. Li, D. F. Zhang, K. Huang, D. C. He, W. P. Long, A Memory-efficient Parallel Routing Lookup Model with Fast Updates, *Computer Communications*, Vol. 38, pp. 60-71, February, 2014.

[8] S. Dharmapurikar, P. Krishnamurthy, D. E. Taylor, Longest Prefix Matching Using Bloom Filters, *IEEE/ACM Transactions on Networking*, Vol. 14, No. 2, pp. 397-409, April, 2006.

[9] W. So, A. Narayanan, D. Oran, Y. G. Wang, Toward Fast NDN Software Forwarding Lookup Engine Based on Hash Tables, *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Austin, TX, 2012, pp. 85-86.

[10] B. Xiao, Y. Hua, Using Parallel Bloom Filters for Multi-attribute Representation on Network Services, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 21, No. 1, pp. 20-32, January, 2010.

[11] Z. Zhou, W. L. Fu, T. Song, Q. Y. Liu, Fast URL Lookup Using Parallel Bloom Filter, *Journal of Electronics*, Vol. 43, No. 9, pp. 1833-1840, September, 2015.

[12] F. Li, F. Y. Chen, J. M. Wu, H. Y. Xie, Fast Longest Prefix Name Lookup for Content-centric Network Forwarding, *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Austin, TX, 2012, pp. 73-74.

[13] W. Quan, C. Q. Xu, A. Vasilakos, J. F. Guan, H. K. Zhang, L. A. Grieco, TB^{2}F: Tree-bitmap and Bloom-filter for a Scalable and Efficient Name Lookup in Content-Centric Networking, *IFIP Networking Conference*, Trondheim, NOR, 2014, pp. 1-9.

[14] L. X. Zhang, D. Estrin, V. Jacobson, B. C. Zhang, *Named Data Networking (NDN) Project*, NDN-0001, October, 2010.

[15] Y. Wang, H. C. Dai, T. Zhang, W. Meng, J. D. Fan, B. Liu, GPU-accelerated Name Lookup with Component Encoding, *Computer Networks*, Vol. 57, No. 16, pp. 3165-3177, November, 2013.

[16] W. Eatherton, G. Varghese, Z. Dittia, Tree Bitmap: Hardware/software IP Lookups with Incremental Updates, *ACM Sigcomm Computer Communication Review*, Vol. 34, No. 2, pp. 97-122, April, 2004.

[17] A. R. Barron, Entropy and the Central Limit Theorem, *The Annals of Probability*, Vol. 14, No. 1, pp. 336-342, January, 1986.

[18] http://urlblacklist.com/ klist.com/.

Biographies

Qingtao Wu studied in East China University of Science and Technology, majored in computer application. He works as a Professor in Henan University of Science and Technology. His research interests include computer security, future Internet security.

Jinrong Yan is a Master degree candidate at the Henan University of Science and Technology. She received her B.S. degree from Henan University of Science and Technology. Her research interests include name lookup and network caching.

Mingchuan Zhang studied in Beijing University of Posts and Telecommunications, majored in Communication and information system. He works as an Associate Professor in Henan University of Science and Technology. His research interests include bio-inspired networks, future Internet.

Junlong Zhu is currently pursuing the Ph.D degree in the computer science and technology from the Beijing University of Posts and Telecommunications. His research interests include large-scale optimization, distributed multi-agent optimization.

Ruijuan Zheng studied in Harbin Engineering University Technology, majored in computer application. She works as an Associate Professor in Henan University of Science and Technology. Her research interests include bio-inspired networks, Internet of Things.