International Journal of Future Generation Communication and Networking
Vol. 9, No. 7 (2016), pp. 87-100
http://dx.doi.org/10.14257/ijfgcn.2016.9.7.09

Steerable Name Lookup based on Classified Prefixes and Scalable One Memory Access Bloom Filter for Named Data Networking

Sheng Huang, Jianghua Xu, Xiaofei Yang, Zhen Wu and Cuicui Niu

Key Laboratory of Optical Communication and Networks, Chongqing University of Posts and Telecommunications, Chongqing, 400065, China
huangs@cqupt.edu.cn; xujiang_hua@163.com;Xiaofyang@163.com; wuzhen_mail@163.com; 916781121@qq.com

Abstract

Name Data Networking (NDN) is an entirely new network architecture, in which data packet forwarding rely on content names instead of fixed-length IP addresses. Under the new naming system, content names are hierarchical structure and have variable length, these pivotal features bring new challenges to meet line speed at large scale. In this paper, we propose Steerable Name Lookup based on Classified Prefixes and Scalable One Memory Access Bloom Filter for Named Data Networking (SNLBF) that prefixes are classified by the number of components and the length of prefix to reduce the size of collection and reduce the expansion rate of the Bloom filter by relieving the rate of collection growth. Then prefixes with different length and component numbers are mapped into different Bloom filter. In order to reduce unnecessary probes, we design the number of component steerable structure (NCS). One memory access Bloom filter and scalable Bloom filter are combined to solve scalability of name lookup engine and speed up name lookup. Because H_3 in scalable Bloom filter is no longer applicable, we design new hash modular arithmetic. Our evaluation results show that SNLBF can achieve high lookup speed and exhibit good scalability to large-scale prefixes table.

Keywords: *Name Data Networking (NDN) Scalable Bloom filter(SBF) One Memory Access,The number of component steerable structure(NCS)*

1. Introduction

NDN is proposed recently as a new network architecture for future network and becomes research hotspot quickly, which directly regards content as core object. NDN no longer concentrate on "where" the content (information) is located, but pay more attentions to the content itself, such as content quality and security[1-2]. At the same time, NDN can solve problems that can't be solved completely in IP network. The name of requested content is treated as the sole basis in the process of requesting data，router forwarding relies upon the longest prefix matching (LPM) of content names. Because of the new naming system, the name are hierarchical, which consists of a series of delimited components and the length of prefix is the number of characters to be contained. For examples, "/uk/co/inventorysoftware" has three components, namely uk, co and inventorysoftware and the length of it is 24. Compared to IP prefix lookup, these emerging features of NDN names bring several unprecedented challenges for LPM name lookup in a practical large scale. These challenges are shown as below:

1) The change of matching granularity. NDN names, unlike IP address, have a series of delimited components. It is different from IP match that LPM in NDN must match prefix at the end of component, rather than at any digit in IP.

2) High update rate. When the Content Stores (CS) around the current node changes, it has to update the forwarding table in the router to ensure efficient routing immediately.

ISSN: 2233-7857 IJFGCN

New content is published and old one is replaced frequently, which must bring high update rate.

3) Variable prefix length. NDN name consists of a series of delimited component which has variable length. It is difficult to organize prefix effectively and needs to consume more computing resources when executing the longest prefix match.

Recently, some researchers have proposed many algorithms on name lookup in order to speed up packet forwarding. These algorithms can be roughly divided into two categories. One is based on Trie, an ordered tree data structure, which related solutions have been conceived. Recently, Wang et al. proposed an effective name component encoding (NCE) to accelerate component matching and developed the State Transition Arrays(STA) to speed up name lookup[3]. However, the whole structure of name still keep Trie, the performance of NCE may degraded with depth of Trie increased. In additional the STA uses pointer to indicate location of the node and the memory cost is relatively high. Zhang et al. proposed Component-Trie whose depth goes down compared to Traditional Character Trie (TCT). But Component-Trie is only allowed to allocate character pointer to store component because component has different length [4].The performance of Component-Trie will decline greatly because more memory access is needed.

Another family of approaches relies on hash functions. CCN_X, achieves LPM as follows: The FIB is a hash table consisting of prefixes [5]. Firstly, the full name is used as key retrieves in the hash table. If the full name is detected, matching is successful. Otherwise we retrieve new name by removing the last component until the name is empty or matching is successful. The number of probe times equals to the number of component to be contained in the prefix at the worst case to make difficult cope with data packet. Wang et .al proposed two-stage Bloom filter, in the first stage it determines the greatest number of components of name and in the second stage it looks up the prefix in a group of Bloom filter[6]. Prefixes are mapped into different Bloom filter depending on forwarding port, which significantly reduced the memory cost. Its performance, however, depends on the distribution of prefix length and the number of ports in the router. With unfavorable prefix distribution and large number of ports, the performance of Name Filter may decrease. Unfortunately, it didn't propose effective measures to solve false forwarding because of the false positive rate of Bloom filter, which causes false packet forwarding and wastes bandwidth resource in the network.

After analyzing existing schemes, we propose Steerable Name Lookup based on Classified Prefixes and Scalable One Memory Access Bloom Filter for Named Data Networking, which can achieve high throughput and high update rate. The main contributions are as follows:

(i) we firstly put forward to classify the prefix by the number of components and length of prefix to degrade the size of the collection and reduce the expansion rate of the Bloom filter by relieving the rate of collection growth. At the same time, prefix with the same number of components and length is stored in each hash table, so each node in hash table can be allocated memory with specific size.

(ii) we design NCS to optimize the number of components when longest prefix matching is performed to avoid unnecessary probe and speed up name lookup. Fortunately NCS is memory-efficient because of high aggregation of the first three components in prefix.

(iii)One memory access Bloom filter and scalable Bloom filter are combined successful. We design new hash modular arithmetic to ensure just one index of the last Bloom filter to be calculated. If the other index is needed, it can be obtained just only by bit shift operation to avoid hash calculation, meanwhile one memory access can greatly speed up name lookup.

The rest paper is organized as follows: Section 2 introduces Steerable Name Lookup based on Classified Prefixes. We describe scalable one memory access Bloom filter in Section 3 and theory analysis for proposed solution are presented in Section 4. In Section 5, we conduct extensive experiment to evaluate the performance of SNLBF, then we

conclude the paper in Section 6.

2. Steerable Name Lookup based on Classified Prefixes

2.1. 3M Prefix Table

The prefix table using in experiment, "3M prefix table", contains 2,739,587 entries. The distribution of components number and the average length of prefix are shown in Table 1. Fig. 1 shows the distribution length of prefix with different component numbers. The 3M prefix table is obtained mainly by following steps:

Step 1. The domain name and URL widely used in academic community are downloaded from DMOZ [7], then domain name is extracted from URL.

Step 2. Hash table is built and chain is used to handle hash conflicts. All domain names are inserted into hash table. If the inserting domain name has already been existed in hash table, the domain name shouldn't be inserted. When all prefixes are completed, all names in hash table are extracted to construct prefix table to realize remove duplicate.

Step 3. The delimiter '.' in the domain name is modified as '/'. We arrange the domain name in reverse order to meet NDN naming convention. For example, "www.baidu.com" is modified as "/com/baidu/www".

Table 1. The Distribution of 3M

Components	1	2	3	4	5	6	7	8	9
Prefixes	64	2104478	544249	83316	7138	310	26	5	1
Average length	3. 69	16. 38	19. 34	22. 81	26. 48	32. 21	34. 7	37	47
Percentage	0	76. 80%	19. 9%	3. 03%	0. 26%	0. 01%	0	0	0

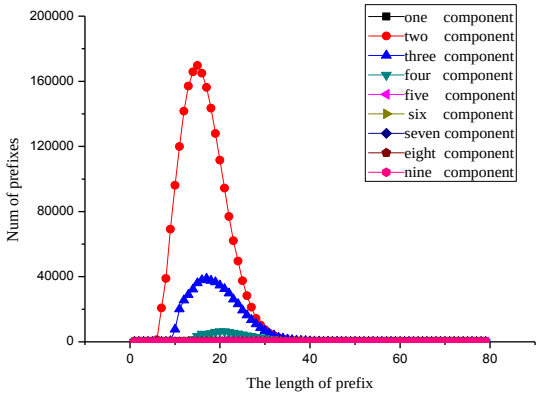

Figure 1. The Distribution Length of Prefix with different Component Numbers

2.2. Classified Prefixes

The general idea of classifying prefixes is that prefixes is classified by components numbers and the length of prefix to reduce the size of the collection.

As shown in Table 1, 2,104,478 prefixes consist of two components and 544,249 prefixes consist of three components, which occupy 76.80% and 19.9% of 3M prefix table respectively. For the prefixes with same components number, the length distribution of prefix approximately is normal distribution, which means that the length of most prefixes

is limited within in a certain range. The number of the prefixes that have one and more than five components is relatively less, so these prefixes are no longer divided and mapped to Bloom filter. For the prefixes whose components numbers between two and five are normal distribution, prefixes with different length in a certain range and component numbers are mapped to different Bloom filter. Then these prefixes that are mapped to different Bloom filter are stored in different hash tables. The number of prefixes that its length beyond a certain range is small, so they don't be mapped to Bloom filter and are stored in the corresponding hash table to reduce the number of Bloom filter. The prefixes that are stored in each hash table have same length so that each node in hash table can be allocated memory with specific size to avoid character pointer, therefore the number of memory access times is reduced to one when lookup is executed in hash node.

Fox example, prefixes with two components whose length is between six and forty-six are mapped to different Bloom filter according to the length of prefix, then they are stored in hash table. The rest of prefixes that length beyond six and forty-six are stored in corresponding hash table directly.

2.3. The Structure of Hash Table and NCS

It is obvious that the Bloom filter can only query whether an element belongs to the set or not, so hash table is needed to store prefix and forwarding port[8-9]. Here, prefixes that mapped to a Bloom filter stored in a hash table have same length. The structure of hash table is presented in Fig. 2. The first part is used to store prefix. The second part represents forwarding port. The third part is the pointer to the next node, and chain is used to solve hash conflicts.

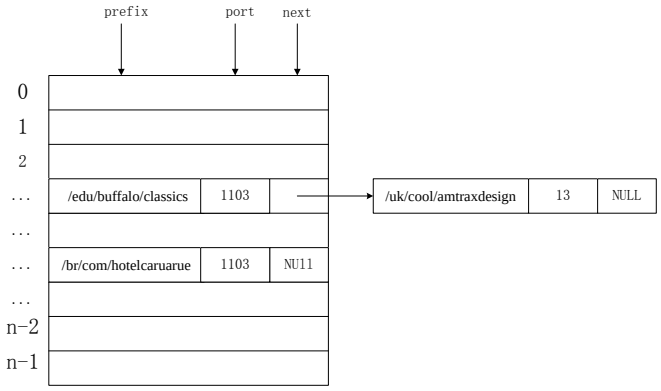

Figure 2. The Structure of Hash Table

Prefixes with the same length is stored in each hash table, so each node in hash table can be allocated memory with specific size avoiding character pointer.

In order to speedup name lookup, prefix that its component numbers is greater than three is used as management object to construct the number of component steerable structure to avoid unnecessary probes. The reason why we choose prefix with greater than three components is that the prefixes that contains two component makes a large proportion to bring difficulty to mange. The prefixes that the number of component is greater than three are mapped to hash table to build component steerable structure. The structure of NCS is presented in Fig. 3. The first part is character pointer. The second part has six bits, because the prefix has nine components at most, after removing the first three components, the number of the rest has six components at most. The third part is pointer to the next node. For example, "/com/itgo/te/sch", "/es/co/blogs/xuven/new" and "/au/edu/nsw/schools/birrongboy/source" are stored in NCS as shown in Fig .3.

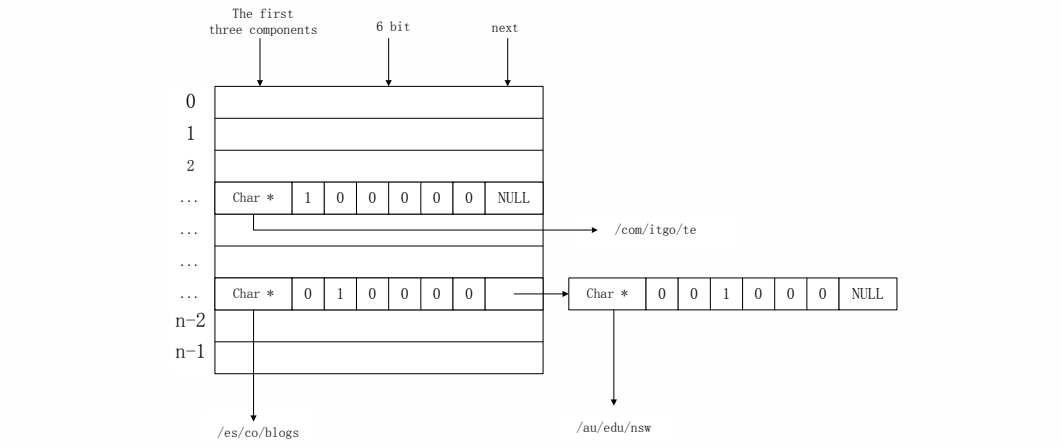

Figure 3. The Structure of NCS

As shown in Fig. 3, the first prefix has four components, so the first bit is set one. The second prefix has five components, so the second bit is set one. The third prefix has six components, so the third bit is set one. When longest prefix matching is executed on the "/es/co/blogs/xuven/new/video/version2.0", it is no longer to find the whole name. The first three components is used as key to obtain index in hash table. According to the index of hash table, we find the string that the first part in the first node points to equals to the first three components "/es/co/blogs" and the second bit is set one, so the first five components is extracted to form new name. If there is no NCS, the number of probe times is three. Luckily, the number of probe times is reduced to once according to NCS.

2.4. The Aggregation of Prefixes in NCS

We define prefix aggregation as $d_i = 1 - \alpha_i / s_i$,s_i represents the number of prefixes whose number of component is greater a_i than three, represents the number of prefixes that the first three component is different in s_i prefixes. The aggregation change is shown in Fig. 4.

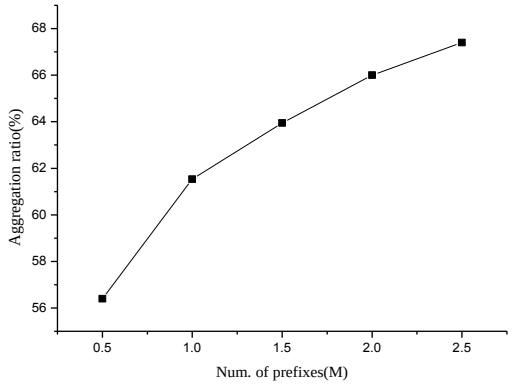

Figure 4. Aggregation of the First Three Components

As shown in Fig. 4, aggregation will be increased along with the size of router table increased, but the growth rate of aggregation becomes slow gradually. Obviously, the first three components can be reused, so the memory cost of NCS should be small.

3. Scalable One Memory Access Bloom Filter

Most of existing Bloom filter with fixed parameters represent static collection, which the vector length of Bloom filter are calculated by maximum false of positive rate tolerated in practical application and the number of hash functions, so the number of prefixes that can be inserted into Bloom filter has upper limit[10]. The Bloom filter is not available when the number of prefixes that has been stored in Bloom filter beyond its upper limit. In practice the routing table needs to update frequently, such as insertion and deletion. When the number of prefixes that has already existed in Bloom filter is greater than its upper limit, the scalability of search engine suffers severely restriction because inserting prefixes into Bloom filter make it unavailable, which no longer suit for routing and forwarding.

Scalable Bloom Filter (SBF) was firstly proposed by Xie to solve scalability [11],the main idea of SBF is that keeping a low false positive rate by adding Bloom filter vectors with double length when necessary. When the number of prefixes existed in Bloom filter has achieved the upper limit, double vector length of Bloom filter are generated to ensure scalability. Because the length of prefix in NDN is variable and no externally imposed upper bound, which brings great difficulty to choose hash transformation matrix H_3, the new hash modular arithmetic is designed to achieve that it only need to calculate one index of the last Bloom filter. If the other index is needed, it can be obtained just by bit shift operation to avoid unnecessary hash calculation. The index of the SBF_i and SBF_{i-1} are originated from i and i-1 bit left shift operation basing on hash value modular arithmetic by m, so the index of SBF_{i-1} can be obtained only by one bit shift operation basing on SBF_i. The details are shown in Algorithm 1.

Algorithm 1 The hash modular arithmetic of Scalable Bloom filter

Step 1. The vector length of initialization Bloom filter SBF_0 is m, which the index of vector from 0 to m-1. The hash value with modular arithmetic by m can be correspond to the index of SBF_0 vector.

Step 2. After the first expansion, the vector length of SBF_1 is twice as long as SBF_0 . The hash value is executed modular arithmetic by m firstly and then one bit left shift operation, which can be correspond to the index of SBF_1 vector.

Step 3. After the i expansion, the vector length of SBF_i is i times as long as SBF_0 . The hash value is executed modular arithmetic by m firstly and then i bit left shift operation, which can be correspond to the index of SBF_i vector.

The corresponding bit of Bloom filter is set one according to Algorithm 1. The index of vector of SBF_{i-1} is $(hashvalue\%m)<<2^{i-1}$ and the index of vector of SBF_i is $(hashvalue\%m)<<2^i$. Obviously, the index of vector of SBF_{i-1} can be obtained that it only need to execute one bit left shift operation based on SBF_i .So only k hash functions need to be calculated to save computation resources.

Compare with hash calculation, memory access wastes much more time. So the number of access memory times should be reduced as far as possible. One memory access of Bloom filter is firstly proposed by Yan [12] .The main idea is that one hash value is used as index of Bloom filter, the rest k-1 hash value are stored in a single word, which can be obtained in one memory operation for all k hash values, thus the memory access times is reduced to one from k.

IP address is used as core object in IP-based network, which consume a relatively low computation resources, but one hash calculation requires at last one traverse of the whole prefix which make the hash computation the dominated consuming task in the name lookup operation in NDN. Algorithm 2 makes only one hash function to be calculated to speed up name lookup:

Algorithm 2 The lookup in one memory scalable Bloom filter

```
1: Procedure Lookup(name, i, SBF)
2:     hash[0….k]      ← 0;
3:             w_s     ←strlen(name);
4:             hashvalue       ←hash(prefix)%m;
5:             hash[0]         ←hashvalue<<i;
6:             word            ←0;
7:           while(i)
 8:            for j     ← 1 to k do
 9:              location      ← hash[j-1]%w_s;
10:             hash[i]        ←(name[location]+hash[j-1])%w;
11:             word           ← word|(1<<hash[j])
12:          end for;
13:          if((SBF_i[hash[0]]&word)!=word)
14:               hash[0]       ←hash[0]>>1;
15:                  i          ← i-1;
16:                else return true;
17:               end   if;
18:       end while
19:   End Procedure
```

Hash[0] is calculated from line 4 to 5 to used as the index of Bloom filter, the line 8 to 11 present that the rest k-1 hash values combined offset and old hash values are mapped to a single word by shifting operation, thus the k-1 hash values are stored in a word. The new $word_1$ is obtained by executing 'and' which are based on word and the word that is stored in the index of hash [0] in Bloom filter. If $word_1$ is equal to word, the match is successful. Otherwise the next new index Hash [0] of Bloom filter can be obtained by simple bit shift operation until matching successful or all Bloom filter are checked .

When the packet arrives, the content name is extracted from the packet to lookup the next forward port. First, the new name that has more than three components is formed according to NCS. Then the Algorithm 2 is executed to test new name whether or not. If the new name has existed in Bloom filter, the next forward is looked in the corresponding hash table.

4. Theory Analysis

For convenience, we summarize the main notions used in this section in Table 2.

Table 2

symbol	The meaning of symbol
k	The number of hash functions in Bloom filter
m	The vector length of initialization Bloom filter
w	The length of word, 32 bits.
h	The number times of extension
l_i	The number of words in SBF_i, $l_i=m_i/w$
n	The number of prefixes in collection
n_0	The number of prefixes can be inserted in SBF_0
n_i	The number of prefixes be inserted in SBF_i
$n_{i\text{-}max}$	The maximum prefixes can be inserted in SBF
t	The number of prefixes in the last Bloom filter

f	The false positive rate of Bloom filter
w_s	The length of prefix
a	Load factor in hash table

4.1. The False Positive Rate of SBF Analysis

The false positive rate of one memory access Bloom filter [11] is show in formula (1)

$$f_i(i,k,n_i)=\sum_{j=0}^{j=n_i} C_{n_i}^{j}(\frac{1}{l_i})^j(1-\frac{1}{l_i})^{n_i-j}(1-(1-\frac{1}{w})^{jk})^k \quad (1)$$

Suppose the dynamic set with n prefixes are mapped to scalable Bloom filter (SBF),which need to expand h times, and t represents the elements in the last Bloom filter:

$$h=\lfloor \log_2(n/n_0+1)\rfloor \quad (2)$$

$$t=n-(2^{\lfloor \log_2(n/n+1)\rfloor}-1)n_0 \quad (3)$$

The maximum number of prefixes in SBF_i (i<h) n_i can be replaced by $2^i n_0$, formula (1) can be translated as follow:

$$f_i(i,k)=\sum_{j=0}^{2^i n_0} C_{2^i n_0}^{j}(\frac{w}{2^i m_0})^j(1-\frac{w}{2^i m_0})^{2^i n_0-j}(1-(1-\frac{1}{w})^{jk})^k \quad (4)$$

So the false positive rate of SBF can be calculated by:

$$f(h,k,n)=1-\prod_{i=0}^{i=h-1}(1-f_i(i,k))(1-f_h(h,k,t)) \quad (5)$$

Miscarriage of justice of Bloom filter only lead to extra lookup in hash table in SNLBF, which don't lead to packet false forwarding. Because miscarriage of justice means that the prefixes out of Bloom filter will be judged in it, lookup will be executed in the corresponding hash table. Luckily, prefixes has been stored in hash table to make matching fail, no forwarding port is looked.

4.2. Time Complexity Analysis

w_s represents the length of prefix, one hash computation has $O(w_s)$ complexity, the average complexity of SBF can be expressed $O(k\times\log_2(n/n_0))$ and SBF which is used in string computation has $O(k\times w_s\times\log_2(n/n_0))$ complexity in average case. One memory access of Bloom filter can reduce the number of memory access times, the average complexity can be reduced to $O((k-1+w_s)\times\log_2(n/n_0))$ in algorithm 2. Meanwhile, the number of access memory times is reduced to one from k in one query.

The length of successful lookup and unsuccessful lookup in hash table can be respectively represented as $s_{nc}\approx 1+\frac{\alpha}{2}$ and $u_{nc}=\alpha+e^{-\alpha}$.The length of lookup is constant which is related with the number of items in hash table. The length of extra lookup in hash table which caused by miscarriage of justice in Bloom filter can be shown as follow:

$$l_{extra}=(\alpha+e^{-\alpha})(1-\prod_{i=0}^{i=h-1}(1-f_i(i,k))(1-f_h(h,k,t))) \quad (6)$$

The average length of lookup in hash table, combining successful and unsuccessful lookup, can be denoted as follows:

$$l_{average}=(1+\frac{\alpha}{2})(\prod_{i=0}^{i=h-1}(1-f_i(i,k))(1-f_h(h,k,t)))+(\alpha+e^{-\alpha})(1-\prod_{i=0}^{i=h-1}(1-f_i(i,k))(1-f_h(h,k,t))) \quad (7)$$

The algorithm proposed in this paper can bring improvement compared to traffic waste in network which caused by false forwarding.

5. Simulation Results and Analysis

In this section, we evaluate the performance of SNLBF and compare it with other name lookup mechanisms in terms of lookup throughput, memory consumption, and update. TCT, CCN_X and BH are used as contrast algorithm.

5.1. Experimental Setup

The name lookup engine is implemented on Linux operating system with the version number Ubuntu13.10. The engine is developed by C++ program language. Relevant hardware configuration is listed in Table 3.

Table 3. Hardware Configuration

Item	Specification
Motherboard	LENOVO (Intel H81 (Lynx Point))
CPU	Intel Core-3300 (4 cores)
RAM	8 G Bytes

5.2. Name Trace

The name traces are used to test the lookup performance of our proposed methods. Two types of name traces, average and heavy workload, are generated to measure the lookup performance that are used to simulate name lookup under average and heavy workload. As for average workload traces, a name is formed by connecting a prefix with suffix which choose from suffix table randomly. As for heavy workload trace, using the same method, but the length of suffix is greater than constant length. So the average length of name in heavy workload trace is greater than the average workload trace. Compare to average workload, the heavy workload will consume much more computation.

5.3. Experimental Result

5.3.1. Memory

The memory cosumption in SNLBF include three parts: (1) SBF (2) Hash table (3) NCS. The comparison with SNLBF, TCT, CCNX and BH is shown as Fig. 5.

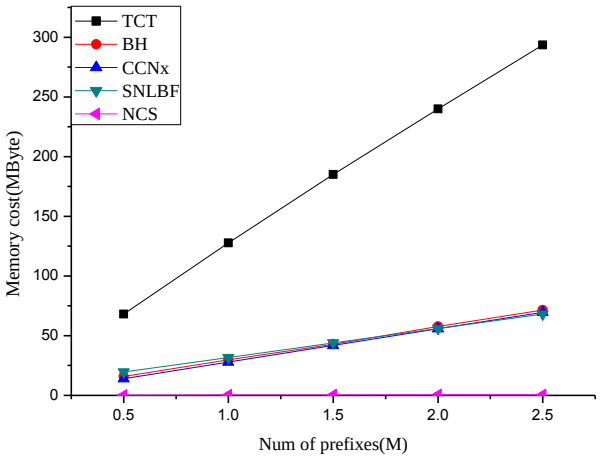

Figure 5. The Memory Consumption of Methods on Different Prefix Table Sizes

Because only one character of prefix is stored in one node of TCT which will cause a great deal of nodes to be generated .Worst, extra memory in one node is used to store other information. Compared to other three methods, TCT consume the most memory. In CCNx, only character pointer is used to point to the name prefix which will cause extra memory that used to store the prefix. In SNLBF, the prefixes stored in most hash tables have the same length so that memory can be allocated with specific size to void character pointer. SNLBF requires a little more memory than BH and CCNx under the case of small scale route table. With the size of route table increased, the three methods consume nearly equal memory between 1.5M and 2.0M, and BH and CCNx need a little more memory compare to SNLBF when the scale of router table achieves 2.5M. The result demonstrates that SNLBF can save memory and has good scalability. NCS is designed to accelerate the name of lookup speed, but NCS only require a little memory because the aggregation of the first of three component of prefixes compressed the memory consumption.

5.3.2. Lookup Speed

First of all, we compare lookup speed on 3M prefix with other four methods under average workload and heavy workload. In order to get the average name lookup speed, we input name with scale correspond to the prefix table each time and get the total execution time, so the average name lookup speed can be obtained. The experiment result is shown as Fig .6.

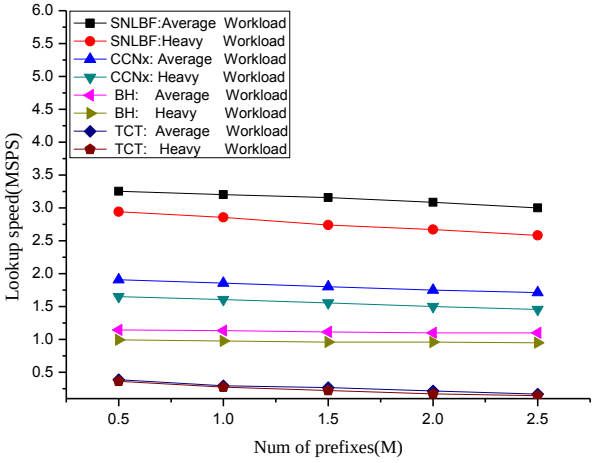

Figure 6. The Speed of Name Lookup on Different Prefix Table Sizes

Fig .6 presents that the average lookup speed slightly decline with the scale of router table increased. TCT has the worst performance because the depth of TCT is very great to lead more memory access. Obviously, the performance of SNLBF is better than BH and CCN_x, because BH must conduct complex hash computing and cope with a number of conflicts. Compared to SNLBF, CCN_x needs many probes and to handle hash conflicts. In SNLBF, the number of probe times in longest prefix matching can be reduced by NCS. SNLBF can speedup name lookup because only one hash function need to calculate in one query and the number of memory access times is reduced greatly.

5.3.3. Update

Fig .7 shows the update rate with the prefix table increased. Obviously, CCN_x has best update performance because the update of CCN_x is relatively simple. SNLBF and BH need to update forwarding plane and data plane. Counter Bloom filter will be update firstly. Counter Bloom filter are mapped to forwarding plane to generate standard Boolean

Bloom filter. In additional, NCS need to be update. But the update rate of SNLBF can achieve one third of the average lookup speed that can meet the actual application. TCT has worst performance because executing name lookup need more memory access and the update process is very complex.

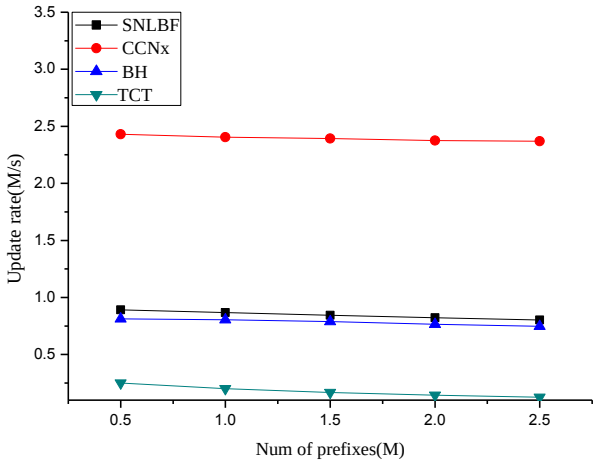

Figure 7. The Update Speed of Methods on Different Prefix Table Sizes

6. Conclusion

In this paper, we propose SNLBF, a fast name lookup method with low memory. We put forward to classifying the prefix by the number of components and length of prefix to degrade the size of the collection and reduce the expansion rate of the Bloom filter by relieving the rate of collection growth. We design NCS to reduce the unnecessary probe. One memory access Bloom filter and scalable Bloom filter are combined successful and we design hash modular arithmetic because H_3 is invalid. Extensive experiments demonstrate that SNLBF can reduce memory cost effectively while guaranteeing high-speed of longest prefix matching in name lookup. In the future, memory consumption should be reduced as soon as possible while keeping high performance. Especially, the number of memory access and hash function should be pay more attentions.

Acknowledgements

This paper is supported by 973 project (2012CB315803), the National Natural Science Foundation of China (61371096, 61171158, 61275077),the Natural Science Foundation Project of CQ CSTC (cstc2013jcyjA40052,cstc2012jjA40060), the Scientific and Technological Research Program of Chongqing Municipal Education Commission (Grant No.KJ130515).

References

[1] L. Zhang, D. Estrin, V. Jacobson and B. Zhang, “Named data networking (ndn) project”, **(2010)**.
[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs and R. L. Braynard, “Networking Named Content”, CoNEXT09, **(2009)**, pp. 1-12.
[3] Y. Wang, K. Q. He, H. C. Dai, W. Meng, J. C. Jiang and B. Liu, “Scalable name lookup in NDN using effective name component encoding”, Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on. IEEE, **(2012)**, pp. 688-697.
[4] T. Zhang, Y. Wang, T. Yang, J. Y. Yu and B. Liu. “NDNBench:A Benchmark for Named Data Networking Lookup”, Next Generation Networking Symposium, **(2013)**, pp. 2152-2157.

[5] CCNx project.http://www.CCNx.org/.
[6] Y. Wang, T. Pan, Z. Mi, H. C. Dai, X. Y. Guo, T. Zhang and B. Liu, "NameFilter: achieving fast name lookup with low memory cost via applying two-stage Bloom filters", IEEE INFOCOM Mini-Conference, **(2013)**.
[7] DMOZ-open directory project [EB/OL], **(2011)**, http://www.dmoz.org/.
[8] S. Dharmapurikar, P. Krishnamurthy, D. E. Taylor, "Longest preifx matching using Bloom filter", IEEE/ACM Trans. Netw, vol. 14, **(2006)**, pp. 397-409.
[9] W. So, A. Narayanan, D. Oran and Y. G. Wang, "Toward fast NDN software forwarding lookup engine based on Hashed tables", ACM ANCS, **(2012)**.
[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors", Communications of the Acm, vol. 13, no. 7, **(1970)**, pp. 422-426.
[11] K. Xie, Y. H. Min, D. F. Zhang, J. G. Wen and G. G. Xie, "A scalable Bloom filter for Membership Queries", IEEE, **(2007)**, pp. 543-547.
[12] Y. Qiao, T. Li and S. Chen, "One memory access Bloom filters and their generalization", INFOCOM, Proceedings IEEE. IEEE, **(2011)**, pp. 1745–1753.

Authors

Sheng Huang, she received his M.S. degree in communication and information system from Huazhong University of Science and Technology in 2003, received his Ph.D. degree in Electrical Circuit and system from Chongqing University in 2008. He is now a professor at School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications. His research interests includeoptical communication system, channel coding and networks.

Jianghua Xu, he is currently pursuing the M.S. degree with the Department of Information and Communication Engineering in Chongqing University of Posts and Telecommunications (CQUPT). His main research interests include name lookup in Name Data Networking and IP lookup.

Xaofei Yang, he is now a vice professor at School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications. His research interests include signal and systems.

Zhen Wu, he is currently pursuing the M.S. degree with the Department of Information and Communication Engineering in Chongqing University of Posts and Telecommunications (CQUPT). His main research intersts include Routing and Forwarding strategy in NDN.

Cuicui Niu, she has got master of engineering degree in chongqing university of posts and telecommunications.Her main research interests include the future Internet architecture and mechanism, efficient transport mechanism in Content-Centric Networking.

