International Journal of Current Trends in Engineering & Technology
ISSN: 2395-3152
Volume: 02, Issue: 01 (JAN-FEB, 2016)

A Fast Name Lookup Method in NDN Based on Hash Coding

Mr. Shuaibo Feng, Assoc. Prof. Mingchuan Zhang, Assoc. Prof. Ruijuan Zheng
School of Engineering and Information Technology
Information Engineering College, Henan University of Science and Technology
Luoyang, Henan Province, China

***Abstract*—In order to improve the storage efficiency and query rate of the name set in NDN network, an efficient name query mechanism based on the element hash encoding is proposed. Firstly, it compressed name set by constructing the element hash value tree of the name set through the incremental hash function. Secondly, the quick name searching is realized through the transition array. Finally, the incremental update mechanism is designed to meet the frequent modify, insert and delete of NDN name. Through experimental analysis can know that the name compression ratio reach to 50% and the query efficiency increase 10%.**

***Keywords* — *longest* name prefix matching, named data network, name component coding, name lookup.**

1. INTRODUCTION

With the increase of information and the expansion of internet application type, the traditional TCP/IP network can't meet the needs of the current users on the network. In recent years, Named data network (NDN) has become the mainly trend in the future Internet. Compared to the traditional IP network, NDN is more concerned about the content, rather than "where" the information is located [1] and it uses the naming of hierarchical structure to replace the traditional IP address [2]. NDN retrieves date based on the name of message, and the route is matched by the longest prefix of the name. Compared with the fixed length IP address, NDN name length does not have upper limit, and the total number of NDN names is much larger than the total number of IP addresses in the network. Therefore, NDN based forwarding table will consume more space than IP address forwarding table. In addition, the NDN name is composed of a number of variable length strings and separators, which increase the complexity of the name lookup. So, in the NDN network, how to realize the efficient storage and quick search of the information name has been widely concerned. The literature [3], [4] use bloom filter to filter the name, the names that have the same number elements are divided into the same name sets. Compared to the hash table, bloom filter has advantages of low storage overhead. But the bloom filter has a very small false alarm rate, which will affect the correctness of the query results. Bloom filter can't meet the rapid query for a large number of data. Yuan et al, proposed a name lookup algorithms based on hash function, using the name as a keyword to the query. The algorithm is difficult to deal with the high speed search of massive data [5]. Literature [6] proposed a multi-step alignment and transfer array to represent the character tree and search, using the GPU parallel processing ability to speed up the search. Although the speed of the search is accelerated, it is limited by the size of the actual memory. When the number of entries that are inserted and updated is large, the update rate is limited. In

literature [7], [8], the flat name is used to replace the IP, but the flat name can't meet the requirements of the name aggregation. Literature [9] proposed an efficient scalable name search algorithm, which compresses the space of strings by the encoding, and through the status of the array to achieve a fast search, updated operations, but it can still improve the encoding way to further compress the string. 2012 Wang et al proposed a name for the component coding scheme [10], through the code distribution mechanism to achieve high efficiency encoding, this scheme improves speed of the prefix matching, but still has the space to be improved in component storage and accelerates the rate of the name lookup. Literature [11], [12] compression storage space by encoding the name, and using the state transition array to quickly find the name list, but its encoding can be further improved. Literature [13] according to the description of the prefix information in the forwarding table to search, and use the hash table to speed up the name update and query rate, although accelerated the name query speed, but the false positive of hash function still affect the correctness of the query. Literature [14] proposed a full name is recognized as a encoding, but it need a new protocol to change encoding table between routers. In this paper, a mechanism of component hash encoding (CHE) that based on the vertical degree of the name tree structure will be proposed. The NDN name is layered with a split operator. At each level, the hash sequence is obtained by using the incremental hash function, and the corresponding state transition array (STA) is constructed to realize the fast searching and updating for the information name.

2. ELEMENT HASH CODING

In this paper, encoding all children nodes of one father's node by using the n element syntax recursive incremental hash function *H (h, s).* Assuming a node N_j, which child node set is N = (n_1, *n2*, ...n_{i-1}, n_i), so a continuous *n* meta hash sequence can be generated by a hash function. That is to say, in all child nodes of N_j, the hash value of the first *i* node is calculated based on the hash value of the first *i-1*node. The hash value of the first child node is calculated based on the hash value of the father node. Assuming the hash value of node N_j is h_j, you can get the hash encoding of all the child nodes in set *N*. As shown in (1):

$$\begin{aligned} h_1 &= H(h_j, n_1) \\ h_2 &= H(h_1, n_2) \\ h_3 &= H(h_2, n_3) \\ &\vdots \\ h_i &= H(h_{i-1}, n_i) \end{aligned} \qquad (1)$$

The hash value of the root node is h0=H(0,scheme), scheme represents the format of the NDN name. As shown in the fig.1, according to above the encoding way we can construct a hash value tree (name component hash tree, NCHT) for the following named collection In Fig.1, the NCHT is composed of 14 elements, representing the 9 data names in Fig.1. Edges that connect a node to its different child nodes formed an encoding collection, known as the original encoding collection. Edge represents the hash value of the child node. For example, "sina" and "yahoo" are children of node 2, edge $h_{2.1}$ represents the hash value of the node "sina", edge $h_{2.2}$ represents the hash value of the node "yahoo". Therefore, $h_{2.1}$=*H* ($h_{0.2}$, *sina), h2.2=H* ($h_{2.1}$, *yahoo)* can be obtained by the incremental hash function. The encoding hash value of the component is determined by the component itself and before a component encoding. Therefore, for many of the original encoding collection we can use a parallel manner to carry out the encoding.

International Journal of Current Trends in Engineering & Technology
ISSN: 2395-3152
Volume: 02, Issue: 01 (JAN-FEB, 2016)

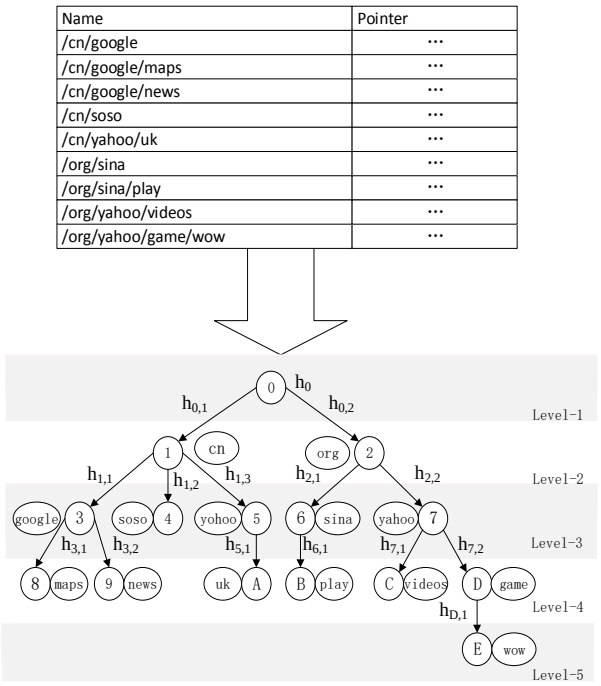

Fig.1. Hash Sequences of Elements

For example, in Level-2 contains two components "cn" and "org", were labeled as node 1 and node 2. Child nodes are {Google, soso, yahoo} and {sina, yahoo}, which are encoded as $\{h_{1,1}, h_{1,2}, h_{1,3}\}$ and $\{h_{2,1}, h_{2,2}\}$. So, the $\{h_{1,1}, h_{1,2}, h_{1,3}\}$ and $\{h_{2,1}, h_{2,2}\}$ can be encoded by a parallel manner. We can find that the same component "*yahoo*" in two different original encoding collections was encoded as different hash values.

3. STATE TRANSITION ARRAY

In this paper, we construct NCHT and achieve the longest prefix matching by using two state transition arrays (Base Array, Transition Array). In this paper, for the first *i* entries in the array *A* are recorded as *A:i*.

3.1. Base Array

The structure of the Base Array is a hexadecimal array; each entry of Base Array is 4 bytes. The first two bytes represent the number of the transition array, which indicates that this element belongs to this transition array. Bits in the back of the Base Array are indicating where the information is stored in the Transition Array. In NCHT, the node number starts from 0. That is, the root node number is 0. Accordingly, the element number of the Base Array is also starting from 0. For example: Base: *i* (*i=0,1,2... n*) represents the state information of the node *i* in NCHT. As shown in fig.2, Base:6 is 0X00030007. So, 0003 represents that the state information of the sixth node in NCHT are stored in Transition 3. And the below bits of Base:6 is a 0007, which represents its information is stored in the seventh entries of the $Transition_3$, and could be expressed as Transition 3:7.

3.2. Transition Array

Transition Array will be set up in each level of the NCHT. And each transition array in different level has a different size. The nodes in each level are stored by the serial number of the state nodes. One transition array is consisted by many segments. Each segment represent one node in the layer. As shown in Fig.2, the first layer of NCHT has only one nodes (the root node), so the $Transition_1$ has only one segment. The length of the segment is not fixed, which is determined by the number of children in the segment. Each segment contains two entries, respectively is an indicator and a status symbol. Each entry takes 8 bytes. The first four bytes in the indicator record the number of status symbols in this segment (the total number of child nodes that the node has). The next four bytes represent the entry pointer. If there is no state in the FIB, PIT, or CS, the entry value is 0. The first value of the state symbol indicates the hash value of the child node. The second value represents the storage location of the child node in the Base Array. All status symbols in one segment constitute a state list for this segment, be recorded as $List_s$ (*s* is the number of nodes that are represented by this segment). For example, $Transiton_3$:1 is an indicator,

International Journal of Current Trends in Engineering & Technology
ISSN: 2395-3152
Volume: 02, Issue: 01 (JAN-FEB, 2016)

its entry has two numbers. 2 indicates that the node has 2 child nodes. 1 indicates that the state node is stored in the first entry of FIB; Transiton$_1$:2 is a status symbol, $h_{0.1}$ represents the hash value of the first child node that belong to the root node. 1 represents the number of the child node in the Base Array.

3.3. Query Process

In Fig.2, given a query name "/org/sina", and its corresponding to the hash encoding is"/$h_{0,0}$/$h_{0,2}$/$h_{2,1}$". Query process as follows:

1) The query process starts from Base:0, Base:0 is the root node of NCHT.
2) Base:0=0X00010001 represent the state information of the root node is stored in Transition$_1$:1
3) Transition$_1$:1 is an indicator, the first four bytes indicates that the root node has two child nodes. That is, node 0 has two status symbols. The hash encoding of the "org" is $h_{0,2}$, which matches the first four bytes of the two status symbol. Find it matches Transition$_1$:3. The last four bytes of Transition$_1$:3 are 2, which correspond to the second elements in the base array Base:2.
4) Base:2=0X00020005, Iterative process to make the hash encoding "/$h_{0,0}$/$h_{0,2}$/$h_{2,1}$" is fully matched. Transition$_3$:7 is an indicator of the node "sina". So far, all components of the name has been matched, no need to turn to the next node to continue the query. Because the last four byte of the Transition$_3$:7 is 6, the pointer points to the sixth entries in the FIB, complete the query.

4. ARRAY GENERATION ALGORITHM

The process of query and insert in the array is shown in algorithm 1. The "name" expresses that the name of input, C1, C2…Cj…Ci expresses the name is divided into i component elements by Decompose (name). LookupLevelj (Cj) indicate the hash value query of the element Cj in the J-th layer array. "S" represents that the state of the current query operation in the transition array T. "S" has three values, "0" indicates that the corresponding hash value is not found in the transition array. "1" indicates that the corresponding hash value is found in the transition array, and the hash value belongs to the states list that are relative to the query node. Due to the limitations of the hash function, there is a situation that may occur in the same transition array, which is hash value of the element C_k in the other state list is the same as the hash value of the query element C_j. In view of this situation, when the "S" value is 2, it is shown that the same hash value is matched in the corresponding transition array, but the hash value that has been found does not belong to the current state list.

Algorithm l: Insert a name to NCHT *Search($name_j$, T)*

1 $(C_1,C_2,\ldots C_i)\leftarrow$Decompose(*name*)
2 for $j\leftarrow i$ to i do // i is the number of components
3 $(h_j,S)\leftarrow H_j\ (h_{j-1},C_j)$ and $lookupLevel_j(C_j)$
4 if $S=1$ then
5 $h_{j+1}\leftarrow Transition(T,S,H_j)$
6 else
7 $List_s\leftarrow Add_H(h_j,C_j)$
8 $(T,S)\leftarrow Add_T(T,S,H_j)$
9 end if
10 end for

Input "*name*" is decomposed by the function *Decompose* (*name*). The function $LookupLevel_j(C_j)$ is used to query the encoding of the elements. The query results can be obtained by the hash value of the element h_j and the value of the current query state S. If S=1, then the query state is transferred to the next element hash value query by *Transition* (T, S, H_j).

International Journal of Current Trends in Engineering & Technology
ISSN: 2395-3152
Volume: 02, Issue: 01 (JAN-FEB, 2016)

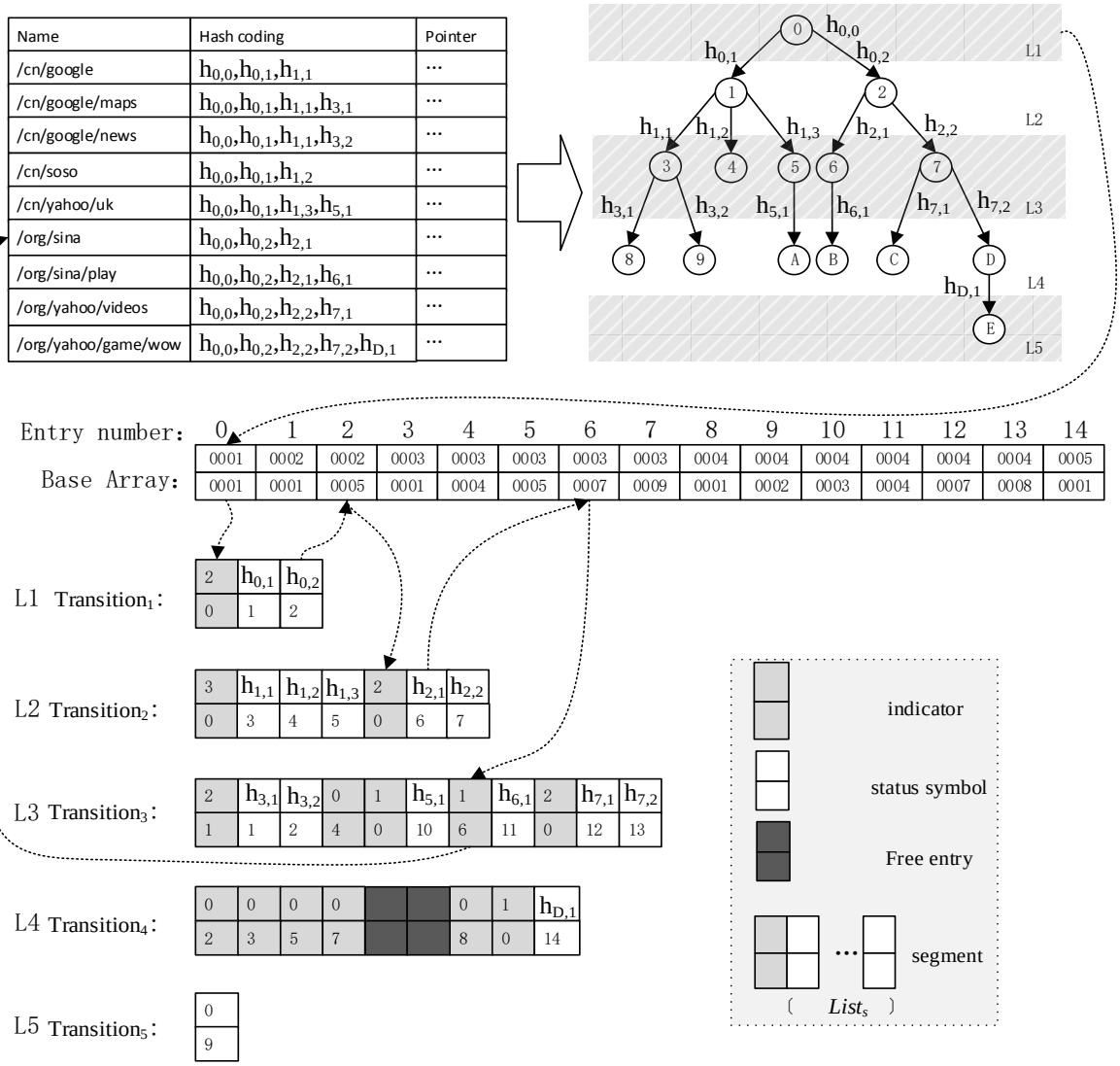

Fig.2.State Transition Array

If S=0 or S=2, indicates that C_j is a new element in this state list, then the hash value will be inserted into the corresponding state list through the function $Add_H(h_j,C_j)$, and create a new state entry in the state array T through $Add_T(T,S,H_j)$, while the state S value becomes 1 The construction of NCHT is achieved by algorithm 2. The delete operation can be realized by amending the entries of the indicator and status symbol. For the node information update operation, it can also be realized by modifying the corresponding state array directly.

Algorithm 2: Building NCHT

```
1 T←NULL
2 for j←1 to K do  // K is the number of Names
3 NCHT-Search (name_j, T)
4 end for
5 return T,h1,h2,...hj
```

5. EXPERIMENT AND PERFORMANCE ANALYSIS

In this section, The memory cost and the query time performance are measured on a PC with an Intel Core 4 Duo CPU of 8GHz and DDR3 SDRAM of 16 GB.CHE mechanism are implemented in C language.

International Journal of Current Trends in Engineering & Technology
ISSN: 2395-3152
Volume: 02, Issue: 01 (JAN-FEB, 2016)

Data from DOMZ [20], CWR [21], Chinaz [22] and ALEXA [23] are used as the input data in experiment. Basic data information is shown in table (1).

Table (1) Number of domains with different components' numbe

Number of components	1	2	3	4	5	6	7	8	9	10	11
CWR	0	10	14537	939485	307109	210684	3771	432	301	49	40
DMOZ	0	0	147345	1376428	431245	48690	3501	153	8	7	0
ALEXA	0	0	1764	239867	153986	77410	1251	96	11	31	331498
Chinaz	0	4	143256	859478	202130	287564	2396	314	117	5	0

The total number of nodes can be expressed as ***node* (*T*) = *edge* (*T*) +1**. The space size of the element hash value tree can be obtained. As in:

$$\begin{aligned} memory &= nodes(T)\times\alpha + edge(T)\times\beta \\ &= nodes(T)\times\alpha + (nodes(T)-1)\times\beta \\ &= nodes(T)\times(\alpha+\beta)-\beta \end{aligned} \qquad (2)$$

In (2), αrepresents the space consumed by each node, β represents the space consumed by each hash value. For the original NCHT (element hash value tree is set up without using the state transition array), each node has at least one pointer to the hash list, a pointer to the forwarding table, and a position number. A hash edge that contains a hash value of a node, a pointer to the next child node, and a number of hash edge list. A pointer consumes 4Byte space, a number consumes 4Byte space, a hash value also consumes 4Byte space. Therefore, α =12, β=12. By the above equation can be the original NCHT will consume **24 × nodes (NCHT) - 12 spaces**. In this paper, we construct the NCHT by two state transition arrays. Each node is represented by an element in Base Array and an indicator in the transition array, a hash edge occupies a status symbol in the transition array. One element in the Base Array consumes 4Byte storage space, one indicator consumes 8Byte storage space, an indicator consumes 8Byte storage space, a status symbol also consumes 8Byte storage space. Therefore, α =12, β=8. By the above equation, the total storage overhead of the nodes tree NCHT is $24\times nodes(NCHT)-8$. So the use of state conversion array to build the NCHT than the original NCHT space is reduced by $1-[20\times nodes(NCHT)-8]/[24\times nodes(NCHT)-12]\approx 16.7\%$. Table (2) shows that the compression of effect on the four domain names.

5.1. Storage Complexity

For an element hash value tree T, the storage space of T is determined by the size of the node and the element edge.

Table (2) Compression ratio of hash sequence

Collection	Total components	Hash chain (Byte)	Tree size (Byte)	Compression Ratio (%)
DMOZ	3124531	10380428	22712462	54.30%
ALEXA	1540737	7823541	16423587	52.37%
CWR	2671263	9716890	21023684	53.78%
Chinaz	2474356	19939764	43459867	54.12%

Total components represent the total number of nodes in the tree. Hash chain represents the bits of the element tree after the hash encoding. Tree size represents the bits of the element tree before the hash encoding. Compression ratio is **1 - HashChain/ TreeSize**. The compression rate in table (2) shows that the method is relatively stable for each data set, which is more than 50%. In order to test the

International Journal of Current Trends in Engineering & Technology
ISSN: 2395-3152
Volume: 02, Issue: 01 (JAN-FEB, 2016)

compression rate of hash encoding on the name set, every time we get the 100K name set from DOMZ to carry out the experiment. From fig.3, we can find that with the increase of the name set, the compression efficiency of the element hash encoding is also increased. Fig.4 shows that the rise of the CHE storage overhead is slower than NCT, and indicates that the CHE is quite effective in both large and small sets.

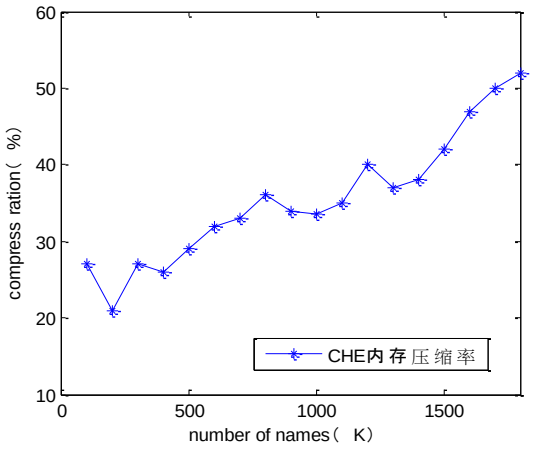

Fig.3. The compression ratio of names in DOMZ

5.2. Query Time Optimization

This paper, through the calculation the average query time of the name set by CHE to prove the optimization effect of CHE on name query. First, input a 200K name set, record query time. Then take this as the foundation, add 200K name set each time, record the time respectively. Calculate the average time of name query at Last. Fig.5 shows the average time of the different domain sets when the number of names is not the same. Analysis shows that different sets of experimental results are different. But, with the increase of the name collection, the query rate tends to be stable. In table (3), we compare the query time between CHE and NCT. Combined with fig.5 we can find that the average search time of CHE is more stable than that of NCT for any data set.

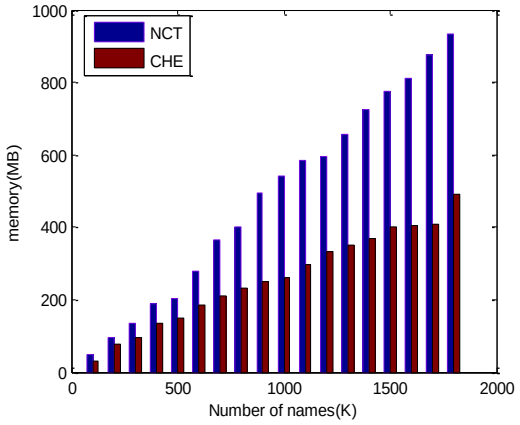

Fig.4.the memory overhead with the change of the name's number in DOMZ

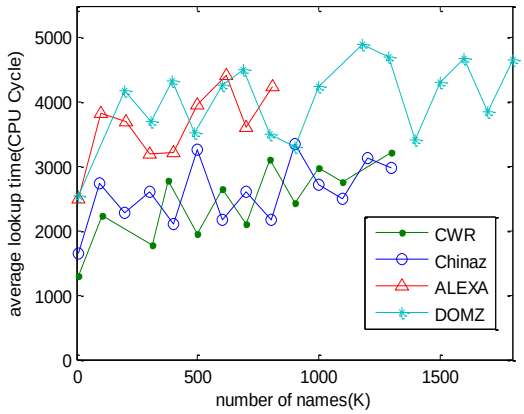

Fig.5. The average query time of different domain sets with the change of name number

Table (3) query time Comparison within NCT & CHE

Data set	Total Domains	NCT Average query time (CPU Cycle)	CHE Average query time (CPU Cycle)	speed up time(s)
CWR	1.5×10^6	18420	3248	5.39
DOMZ	2×10^6	29414	4284	7.16
Chinaz	1.5×10^6	10213	2894	4.78
ALEXA	1×10^6	27314	4179	7.63

It can be found that the average search time of CHE is 2500 ~ 4650 CPU cycles (CPU frequency is 2.8GHz), which is equal to the time of 904ns ~ 1054ns. Comparison shows that CHE can be faster for the longest prefix matching than NCT, the query rate increased by about 10% compared to NCT.

6. CONCLUSION

In order to solve the problem of name storage and name query in NDN network, this paper proposes a new method based on hash coding. In this method, the element hash value tree is constructed by using the incremental hash function to realize the compression of the information name. The fast query and update of the information name is realized through the state transition array. The simulation results show that the method has good performance. In order to further improve the efficiency of the name storage, how to effectively use the free entries in the Basic Array and the Transition Array will be the focus of the next step.

REFERENCE

[1] Ekambaram V, SivalingamI K M, "Interest flooding reduction in Content Centric Networks", High Performance Switching and Routing, 2013 IEEE 14th International Conference on, pp. 205-210, 2013.

[2] Jacobson V, Smetters D K, "Networking named content", Proceedings of the 5th international conference on Emerging networking experiments and technologies, pp: 1-12, 2009.

[3] Wang Y, Pan T, MI Z, "Name Filter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters", INFOCOM, 2013 Proceedings IEEE, pp. 95-99, 2013.

[4] Quan W, Xu C, Vasilakos A, "TB2F: Tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking", Networking Conference, pp. 1-9, 2014.

[5] Yuan H, Song T, Crowley P, "Scalable NDN forwarding: Concepts, issues and principles", Computer Communications and Networks, 2012 21st International Conference on, pp. 1-9, 2012.

[6] Wang Y, Zu Y, "Wire Speed Name Lookup: A GPU-based Approach", NSDI, pp. 199-212, 2013.

[7] Wang Y, He K, Dai H, "Scalable name lookup in NDN using effective name component encoding", Distributed Computing Systems, 2012 IEEE 32nd International Conference on, pp.688-697,2012.

[8] Singla A, Godfrey P, Fall K, "Scalable routing on flat names", Proceedings of the 6th International Conference, 2010.

[9] Jain S, Chen Y, Zhang Z L, "Viro: A scalable, robust and namespace independent virtual id routing for future networks", INFOCOM, 2011 Proceedings IEEE, pp.2381-2389, 2011.

[8] Wang Y, Dai H, Zhang T, "GPU-accelerated name lookup with component encoding", Computer Networks, vol-57(16), pp.3165-3177,2013.

[9] Zhou Z, Song T, Jia Y, "high-performance url lookup engine for url filtering systems", 2010 IEEE International Conference on, pp.1-5, 2010.

[10] Michel B S, Nikoloudakis K, Reiher P, "URL forwarding and compression in adaptive web caching", Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings, pp. 670-678.

[11] Wang Y, Xu B, Tai D, "Fast name lookup for Named Data Networking", Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of, pp.198-207, 2014.

[12] Yu Y, Gu D, "The resource efficient forwarding in the content centric network", Springer Berlin Heidelberg, pp.66-77, 2011.

[13] http://www.chinarank.org.cn.

[14] http://www.alexa.corn.

[15] http://www.dnloz.ore.

[16] http://www.chinaz.cor.

AUTHOR'S PROFILE

(1) Shuaibo Feng – Shuaibo Feng was born in Henan Province, PRC in August 1987. He studied in Henan University of Science and Technology from 2013 to now. His majored research area is The next generation of internet.

(2)Mingchuan Zhang – Mingchuan Zhang was born in Henan Province, PRC in May 1977. He studied in Beijing University of Posts and Telecommunications (Beijing, PRC) from Mar 2011 to Mar 2014, majored in computer application and earned a Doctor of Engineering Degree in three years' time. He works as an Associate Professor in Henan University of Science and Technology from Mar. 2005 to now. His research interests include ad hoc network, Internet of Things, cognitive network and future Internet technology.

(3)Ruijuan Zheng- Ruijuan Zheng was born in Henan Province, PRC in Mar 1980. She studied in Harbin Engineering University Technology (Harbin, PRC) from Mar 2005 to Mar 2008, majored in computer application and earned a Doctor of Engineering Degree in three year's time. She works as an Associate Professor in Henan University of Science and Technology from Mar 2008 to now. Her research interests include bio-inspired networks, Internet of Things, future Internet and computer security.