A Fast Name Lookup Method in NDN Based on Hash Coding

Shuaibo Feng[1], Mingchuan Zhang[1*], Ruijuan Zheng[1], Qingtao Wu[1,2]

[1]Information Engineering College, Henan University of Science and Technology,

Luoyang 471023, Henan Province, China

[2]School of Engineering and Information Technology,

University of New South Wales, Canberra ACT 2600, Australia

fen_bzx@163.com, zhang_mch@haust.edu.cn*, rjwo@163.com, lywxr2007@haust.edu.cn

Keywords: Named Data Network; Hash Coding, Longest Name Prefix Matching; Name Component Coding;

Abstract. Named Data Network (hereinafter called NDN), as one new content-centered network architecture, which routes and forwards the data in accordance with their names. And it satisfies the users' demands for efficient access to mass and heterogeneous information in the network. However, for the data name, their structures are usually hierarchical and lengths variable, then the quick name-searching match becomes a very big challenge. In this essay, Hash Coding Name-searching System is introduced and proposed. In particular, the data name is compressed through the hash function, the quick name-searching is realized through the transition array, finally, the incremental update mechanism is designed to meet the frequent modification, insertion and delete of NDN name. Through experimental analysis we can know that the method will make the name compression ratio reach to 50% at least and the query efficiency increase 10% almost.

Introduction

With the increase of information and the expansion of internet application type, the traditional IP packet exchange mechanism as the core operation of the internet can't meet the current requirements of the network. In recent years, Named data network (NDN) has become the main trend in the future Internet. Different from the traditional TCP/IP network, NDN separates the data storage location with the data itself, and the needs of whole network from the host to the content oriented. Compared to the traditional IP network, NDN is more concerned about the content, rather than "where" the information is located [1]. The advantage is to improve the network performance, which more conducive to share the network's resources.

Different from the traditional IP network, NDN network uses the naming of hierarchical structure to replace the traditional IP address [2]. The original IP address retrieval method has not adapted to the new generation of NDN network, so it is a key task to study the new method of name lookup in NDN network.

The traditional IP address is composed of 32 or 128 digits, with a fixed length. Since NDN name prefixes are some variable length strings, so NDN's name will occupy more space, and the number of NDN's name is also far more than IP address in the network, so the NDN's forwarding table consumes more space than IP's forwarding table. In NDN, the hierarchical structure is adopted to name the information, the name is composed of many variable length strings and separators, and the length of the name is no upper limit, which increased the complexity of the name lookup, make the prefix matching consumed more time and the update operation is more complex. The original search algorithm will take a certain amount of time in determining the length of the name, result in the consumption of time. So it is great challenge to compress the name space of NDN, realize the fast search and the fast update of name.

According to the particularity of the name's lookup in NDN, 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. Literature [5] proposed that the whole name of the information is stored in hash encoding. Literature [6]

proposed route search mode based on hash encoding, mainly used the incremental hash function, encoding from the depth of the tree structure of the name. 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 to construct the state transfer array (STA) for the hash method to simplify the query and update operation, to achieve a rapid longest prefix match.

Element Hash coding

In this paper, we can definition that when a state node S_i is connected to a child node by an element edge C_i, so called $C_i \in S_i$, and definition a given node is connected to its children by a collection of edges, known as the original encoding collection. The hash values of each element in the original encoding collection are not equal, and there is no overlap. in a tree structure. Then using the *n* element syntax recursive incremental hash function ***H(h,s)*** to hash encode for all children nodes of one father's node. Assuming a node N_j, which child node set is $N = (n_1, n_2, n_3 \mathbf{L}\, n_{i-1}, n_i)$, so a continuous n meta hash sequence associated with each child node can be generated by a hash function. That is to say, after the hash function, the hash value of the child nodes *i* is associated with the hash value of the child nodes *i-1*.

Assuming the hash value of the nodes N_j is h_j, So we can get the hash encoding series of all the children of the node N_j. As shown below Eq 1:

$$\begin{aligned} h_1 &= H(h_j, n_1) \\ h_2 &= H(h_1, n_2) \\ &\mathbb{M} \\ h_i &= H(h_{i-1}, n_i) \end{aligned} \tag{1}$$

Given root node hash value is $h_{0,0}$, according to above the encoding way we can construct a hash value tree for the following Named collection. Is shown in Fig.1.

Name	Pointer
/cn/google	…
/cn/google/maps	…
/cn/google/news	…
/cn/soso	…
/cn/yahoo/uk	…
/org/sina	…
/org/sina/play	…
/org/yahoo/videos	…
/org/yahoo/game/wow	…

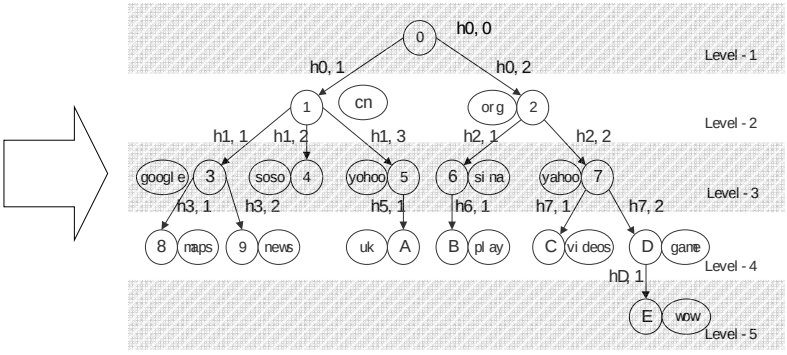

Fig.1. Hash sequences of elements

Fig.1 shows the nine names from 14 nodes to form a NCHT. By definition of the original encoding collection, the hash encoding that connects a given node with its different children nodes should be different. These elements constitute the original encoding set. For example, "sina" and "yahoo" belong to node 2, which were compiled for <sina, $h_{2.1}$> and <yahoo,$h_{2.2}$> respectively, so {sina, yahoo} is the original encoding collection of the node 2.

Since the components encoding are only related to the component itself and its corresponding node, therefore each of the original coding can be independently encoded. Such as, In Level-2 contains two components "cn" and "org", start at node 1 and end at node 2. The children nodes are {sina, soso, yahoo} and {google, yahoo}, be encoded as <google, $h_{1.1}$>,<soso, $h_{1.2}$>,<yahoo,$h_{1.3}$> and <sina,$h_{2.1}$>,<yahoo,$h_{2.2}$>. As can be seen, the same component yahoo is encoded for different hash values in two different original encoding collection （Being Encoded <yahoo,$h_{1.3}$> in the original encoding collection of the first node. Being Encoded <yahoo,$h_{2.2}$> in the original encoding collection

of the second node) . So, given an element with no node information is unable to determine its corresponding node.

Because the NDN name element is separated by a split operator, we can divide the given element to the corresponding level. The Parallel hierarchical search can be performed when executing a query operation.

State transition array. In this paper, we construct NCHT and achieve the longest prefix matching by using two state transition arrays, respectively are Base Array, Transition Array. Transition Array is consisted by different sizes entries in each level arrays. Setting up the corresponding transition array in each level. For example $Transition_2$ is the second level transition array of the NCHT. The nodes in each level are stored by the serial number of the state nodes. For convenience, suppose all the arrays discussed in this paper are indexed from one and we refer the i-th entry of array A as A:i.

The entry of Base Array is 4 bytes. In order to express the intuition, the entries of the Base Array start from 0, 0-th entries are represented by the root node C_0, next turn shows each state node in NCHT. For example: $Base:i$ ($i = 0,1,2$ **L L** n) shows the state i of NCHT. The first two bits of the Base Array entries are represented that whether the information are stored in the Transition Array. 01 represents the information are stored in $Transition_1$. 02 represent the information stored in $Transition_2$, and so on. Bits in the back of the item are expressed where the information are stored in the Transition Array. Is shown in Fig. 2: $Base:4$ is 0X00030004, that represents the state information of the fourth nodes in NCHT are stored in $Transition_3$. And the below bits of $Base:4$ is 4, which represents its information is stored in the fourth entries of the $Transition_3$, and could be expressed as $Transition_3:4$.

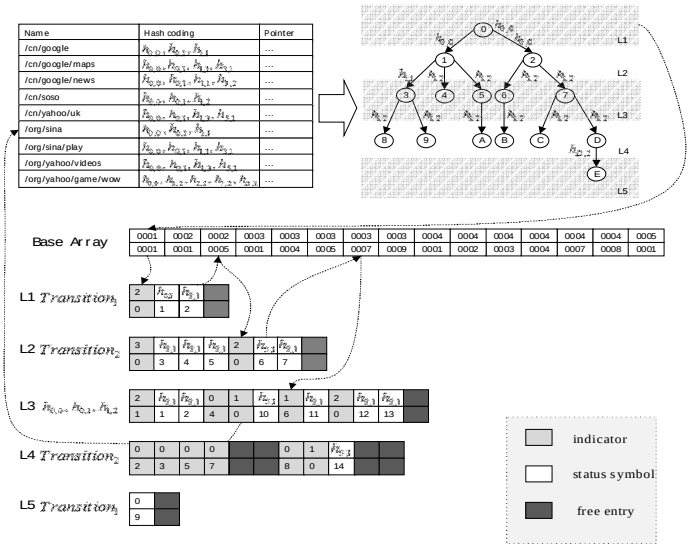

Fig. 2. State transition array

There are two type entries in the Transition Array, indicator and status symbol. The first number of the indicator represent the number of state transition (That is, how many children nodes that the state node has), the second number is the entry pointer, if this state does not exist in FIB, PIT and CS, the entry pointer is 0. The first value of the status symbol is represented a hash encoding that generated by the component hash encoding. The second value represents the storage location of the next state in the Base Array. For example , $Transition_3:1$ is an indicator, The first value represents the next states that connected to the third state nodes have two states, and the second value indicates that the information is stored in the first entries in the FIB.

For example, given a query name "/com/google", and its corresponding to the hash encoding is"$/h_{0,0}/h_{0,2}/h_{2,1}$". Query process is shown in fig 2 and as follows:

I. The query process starts from $Base:0$, $Base:0$ is the root node of NCHT.

II. $Base:0$ =0X00010001 represent the state information of the root node is stored in $Transition_1:1$.

III. $Transition_1:1$ is an indicator, the number on the top of the entry indicates that the root node has two child nodes, that is, the state 0 corresponding to the two state entries. After binary search for these two state entries, Hash encoding match $Transition_1:2$. And because the second value of $Transition_1:2$ is 2, corresponding to the second entries of the Base Array.

IV. $Base:2$ =0X00020005, the iterative process make the hash encoding sequence is fully matched, and the entry pointer point to the sixth entries in the FIB, then complete the query process.

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. Data from DOMZ[7], CWR[8], Chinaz[9]and ALEXA[10] are used as the input data of the experiment.

Storage complexity. The number of nodes in the tree structure is equal to the number of edges plus 1. So the total number of nodes in NCHT can be expressed as $nodes(T) = edges(T) + 1$. Because $edges(T)$ is equal to $hashes(T)$, the size of the storage space can be obtained. Is shown in below Eq 2:

$$\begin{aligned} memory &= nodes(T) \times a + hashes(T) \times b \\ &= nodes(T) \times a + (nodes(T) - 1) \times b \\ &= nodes(T) \times (a + b) - b \end{aligned} \tag{2}$$

In the original NCHT, each node contains two pointers and a list at least, which is a pointer to a hash table, a pointer to a matching entry, a position number, and a list that contain it's hash value and the element edge of the next node is connected. Each pointer is 4Byte, and the hash value is 4Byte. Here a is 12, b also is 12. So the storage overhead of the hash tree can be obtained by the above equation is $24 \times nodes(NCHT) - 12$.

In this paper, a state transfer array is used to construct the NCHT. A entry in the Base Array and an item in the Transfer Array represents a node. Each hash edge needs an item in the array. Each item of the Base Array is 4Byte, and a single entry for the Transfer Array is required for 8Byte.From the above equation, the total cost is $20 \times nodes(NCHT) - 8$. So Building NCHT with a state transition array occupy less space than the original NCHT, the percentage reduction is $1 - [20 \times nodes(NCHT) - 8] / [24 \times nodes(NCHT) - 12] \approx 16.7\%$. At the same time, the elements of the hash encoding also compressed the size of the element string.

Table. 1 The compression ratio of hash sequence

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

Table 1 shows the compression effect of the four domain name sets. Here, total components refer to the total number of nodes in the tree. Tree size represents that byte before compression NCHT. Hash chain represents that byte after compression NCHT. The compression ratio is . We can learn that the compression rate is relatively stable from the Table 1. This can Show that many names use the same elements and path prefix.

Fig 3 shows the relation between the number of domain and the storage overhead of CHE. We can find that the compression efficiency of CHE increases with the increase the number of names. 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.

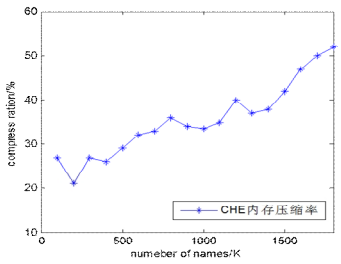

Fig. 3. The compression ratio of names in DOMZ

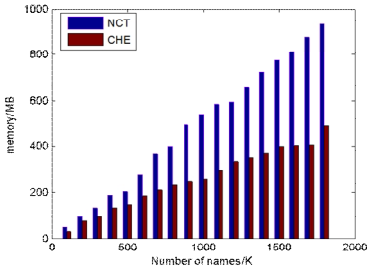

Fig. 4. The memory overhead with the change of the name's number in DOMZ

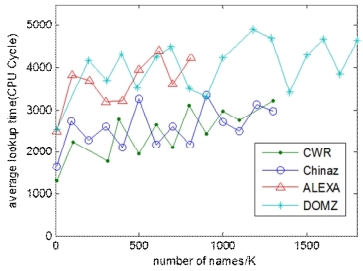

Fig. 5. The average query time of different domain sets

Query Time Optimization. CHE mechanism can optimize the query rate of the name. Each time we input the 200K's name, and get the total execution time, then calculate the average time of the query. 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.

Conclusions

With the increasing popularity of Internet applications, the flow of information on the Internet is increasing. NDN as a new type of network architecture, it uses the data name as the only identification in the whole network. It directly searches and forwards data according to data name. In order to improve the efficiency of data retrieval, this paper proposes a name query method based on hash encoding. On the one hand, compressing space occupied by information storage by hash function, on the other hand, to do search, insert, modify, and delete operations quickly for names by the state transition array. Finally, through the experiment, the effect of the compression and the searching rate is improved obviously. But how to make full use of the free space in the array is the focus in the next research.

Acknowledgement

This work is partially supported by the National Natural Science Foundation of China (NSFC) under Grant no.U1404611 and 61370221, in part by Program for Science & Technology Innovative Research Team in University of Henan Province under Grant no. 14IRTSTHN021.

References

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

[2] L. Zhang L, D. Estrin, J. Burke J. Named data networking (ndn) project,J. Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC, 2010.

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

[4]Wang Y, Pan T, Mi Z, et al. NameFilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters[C]//INFOCOM, 2013 Proceedings IEEE. IEEE, 2013: 95-99.

[5] Z. Zhou, T. Song T, Y. Jia. A high-performance url lookup engine for url filtering systems,C. Communications (ICC), 2010 IEEE International Conference on. IEEE, 2010: 1-5.

[6] Chuanzhen Du, Julong Lan, Ming Tian. Content routing lookup mechanism based on hash coding, J. Application Research of Computers. In Chinese. 31(2014) 3081-3086.

[7] Information on http://www.chinarank.org.cn.

[8] Information on http://www.alexa.corn.

[9] Information on http://www.dnloz.ore.
[10] Information on http://www.chinaz.corn.