Fast Name Lookup based on Component State Transfer in Content Centric Networking

Xiaofei Yang, Jianghua Xu, Sheng Huang and Sijun Liu

Key Laboratory of Optical Communication and Networks, Chongqing University of Posts and Telecommunications, Chongqing,400065, China
xiaofyang@163.com; xujiang_hua@163.com;huangs@cqupt.edu.cn; 815818742@qq.com

Abstract

Content Centric Networking (CCN) is treated as one of the most representatives of the future Internet architecture structure, in which has attracted attention. Interest Packet forwarding relies on content name instead of IP address, the structure of packet name occur hierarchical and the length of name is unlimited, these pivotal features bring new challenges to meet high speed at large scale. In this paper, Fast Name Lookup based on Component State Transfer in Content Centric Networking(FNLCST) is proposed. Hash function and component coding are combined to construct virtual component Trie, then every component is compiled into unique code. In order to improve performance further, components in the same level in the virtual component Trie are stored to different hash tables according to their lengths, so the great number of children nodes can be dispersed by hash function. In addition, the number of memory access times can be reduced because each node in hash table can be allocated memory with specific size to avoid character pointer. Simulation results show FNLCST can achieve high lookup speed and exhibit good scalability to large-scale prefixes table.

Keywords: CCN, LPM, Virtual Component Trie, Hash Table, Memory Consumption, Interest Packet

1. Introduction

TCP/IP protocol suite as the core of the Internet has evolved over fifty years, which has achieved overwhelming success. The purpose of the original Internet interconnection is used to share expensive hardware resources. Usually two computers are connected according to IP address. However, with the Internet scale and application mode changing, the shortcoming of original design in terms of security, mobility, expansibility and other aspects are exposed. Especially, with the high rapid application for the main purpose of “share information” development, current Internet architecture is facing unprecedented challenges [1]. People began to realize that the current network cannot meet the needs in the future.

CCN is proposed recently as a new network architecture, which directly treat content as core object and no longer concentrates on “where” the content is located, but pays more attentions to the content itself, such as content quality and security[2-3]. In CCN, there are mainly two packets: Interest Packet and Date Packet**.** The forwarding architecture mainly contains three tables: CS(Content Store), PIT(Pending Information Table) and FIB(Forwarding Information Base). CS is used to store content that can serves for identical future request. PIT is used to record the arriving port of request that have been forwarded to upstream but has not responded with data packet. FIB is similar to the conventional forwarding table that contains name prefixes and next forwarding port.

In router forwarding, the name of requested Interest Packet is treated as the sole basis in the process of requesting data. Router forwarding relies upon the Longest Prefix

Matching (LPM) of Interest Packet name. At the same time, CCN can solve many problems that can't be solved completely in IP network, such as exhaustion of IP addresses, scalability, mobility and security and so on. Because of the new naming system, the name of Request Interest Packet that consists of a series of delimited components are hierarchical. The length of prefix is the number of characters to be contained. For example, "/sports/rambler/news/yahoo" has four components, namely "sports", "rambler" ,"news" and "yahoo", its length is 26. Comparing name LPM with IP LPM lookup, these emerging features of CCN names bring difficulty to name LPM lookup in a practical large scale. These difficulty are shown as below:

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

2) High Speed. Current routers are treated as core equipment with high bandwidth, name lookup must have high speed to support.

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

4) Variable prefix length. CCN name consists of series of delimited components which have variable length. Prefixes should be stored with effective organize to reduce memory consumption as soon as possible.

Recently, some researchers have proposed many algorithms on name lookup in order to speedup packet forwarding. These algorithms can be roughly divided into two categories. One is based on Trie, an ordered tree data structure, which many related solutions have been conceived [4]. Recently, Wang *et al*. proposed an effective name component encoding (NCE) to accelerate component matching, State Transition Arrays(STA) is used to speedup name lookup[5]. However, encoding resolve conflicts is too complicated. Zhang *et al*. proposed Component Trie whose depth goes down as for Character Trie. But Component-Trie is only allowed to allocate char pointer to point component because components have different number of character [s6]. The performance of Component-Trie will decline greatly because the number of memory times is increasing and the number of average children is too big because of high granularity.

Another family of approaches relies on hash function. Wang proposed two-stage Bloom filter called NameFilter, in the first stage it determines the greatest number of components of name, it looks up the prefix in a group of Bloom filter in the second stage[7]. 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 NameFilter may decrease. Unfortunately, it didn't propose effective measures to solve false forwarding because of the false positive rate of Bloom filter, which wastes bandwidth resource in the network. In addition, Dai proposed BFAST that combines PIT, CS with FIB, which use pointer to point different table. It needs too many Count Bloom Filters to balance hash load, but it doesn't reflect hash load accurately and consumes too much memory.

After analyzing existing schemes, Fast Name Lookup based on Component State Transfer in Content Centric Networking is proposed, which can achieve high throughput and update rate. The main contributions are as follows:

1) Hash function and component coding are combined to construct virtual component Trie, which take advantages of the quick speed of hash and the low depth of component Trie. Different components in the same level in the component Trie are compiled into different codes, so nodes can be reused if it appear more than once. Therefore, the number of stored nodes can be reduced to save memory.

2) Components in the same layer are mapped into different hash tables according to length.

Each node can be allocated memory with specific size to avoid character pointer and to reduce memory access because components that are stored in each hash table have same length.

The rest paper is organized as follows: Section 2 introduces Fast Name based on Component State Transfer in Content Centric Networking. Theory analysis for proposed solution are presented in Section 3. In Section 4, we conduct extensive experiment to evaluate the performance of FNLCST, then we conclude the paper in Section 5.

2. Fast Name Lookup based on Component State Transfer

2.1. Virtual Component Trie and Code Allocation

前缀	端口
/com/mun	53
/com/tumblr/fund	13
/ru/mun/bogoro	23
/org/scz	3
/cz/tumblr/ajaz	14
/cz/tumblr/bogoro	12

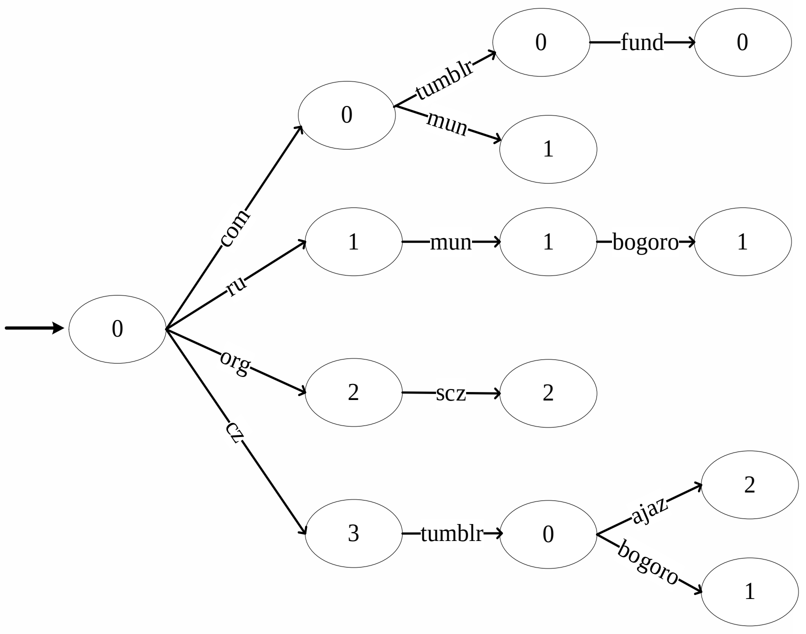

Figure 1. Component Trie

Prefixes are divided into many components according to delimiter boundary. For example, the prefix "/com/itgo/telitnetwork" is decomposed into "com", "itgo", "telitnetwork", then components are used to construct virtual component Trie. Components in the same level are compiled into unique code, namely even if components that come from different parent nodes still have same code. In order to save code, code management lists are created at every level to store code that are allocated for the deleted node. For example, the example of constructing component Trie and code allocation is shown as Figure 1.

Obviously, the depth of component Trie can be dropped to upgrade the performance. There are no two identical components in the first level because of multi-branch Trie high polymeric, but components are possible reused in the second level and above. For example, 'mun' appear twice which comes from the parent of 'com' and 'ru' , but it only has one code '1'.

2.2. Data Structure

Component Trie is lack of practicality because the process of construct waste too much time. In this paper, we propose that hash and encoding are combined to keep component Trie in the logic to reduce unnecessary node pointers between parent and child[9]. In addition, components in the same level are divided according the length of component, so the children nodes can be dispersed to speedup lookup. Component and code are connected by hash, each hash table stores components with same length. Nodes in hash table can be allocated memory with specific size to avoid char pointer because components that are stored in the node has same length, so the number of memory access times can be reduced. The hash node mainly has four parts: 1) component; 2) code; 3) parent node and forwarding port; 4) next pointer. The structure of parent node and forwarding port is shown as follows in Figure 2:

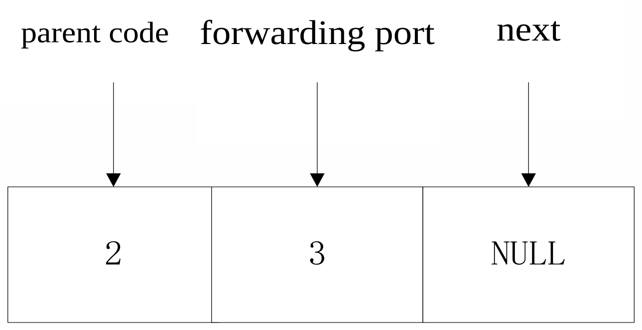

Figure 2. The Structure of Parent and Forwarding Port

The left part is the parent code, the middle part is the forwarding port. The right part is used to point to the next node. If the current component is the last component in the name, the middle part stores the forwarding port, otherwise -1 is stored to indicate invalid port. The structure of hash node is shown in Figure 3.

Components with the length of three in the second layer in the Figure 1 are used to illustrate the structure of hash table. There is only one “scz” which come from “/org/scz”, so the third part directly store the information of parent and forwarding port. The component “mun” appear twice which come from “/com/mun” and “/ru/mun/bogoro”, so the third part become list to store two nodes.

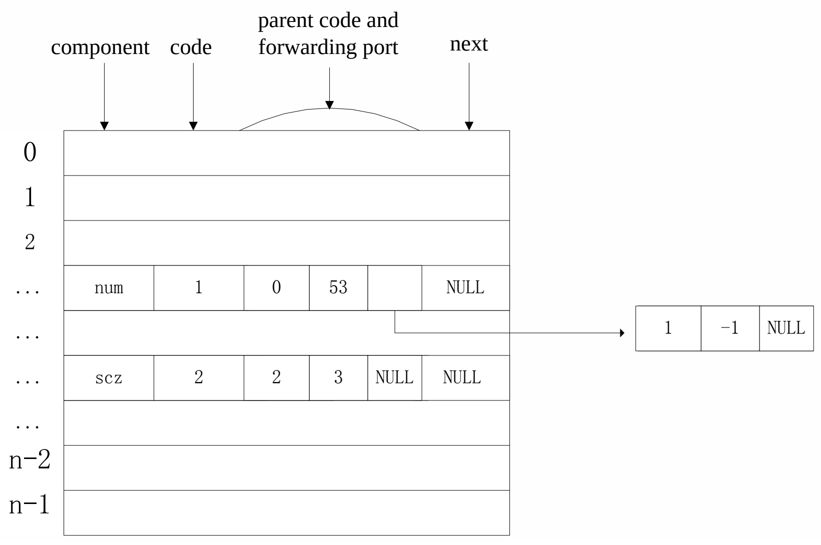

Figure 3. The Structure of Hash Table

2.3. The Process of Router Table Initialization

Prefixes are inserted into the router table until all finished to construct the router table. Specific steps are as follows:

Step 1: Input prefix and calculate the number of components.
Step 2: Handle component successively until all components finished. Check current component in the corresponding hash table nor not. If yes, go to Step3; otherwise go to Step4.
Step 3: Judging the code list of parent node contain the current parent code or not. If yes, set parent code as the code of current node. Otherwise add current parent code to the list and set parent code as current node . Then go to step5.
Step 4: Add current component into the corresponding hash table and set code of current component as current layer status, then current layer status plus one . set parent code as the code of current node .Then go to step5.
Step 5: If the current component is the last, forwarding port be set the port of prefix. Otherwise, -1 be set to indicate invalid port. Then go to step2.

2.4. The Process of Name Lookup

The number of components is calculated firstly when name is coming. The process of longest prefix matching start from the first component, the detail process is described in

Algorithm 1.

Algorithm 1: Longest Prefix Match Name Lookup

```
1:    procedure Lookup(name)
2:    k←sumofcomponent(prefix x);//calculate the number of component
3:    (C_1,C_2,C_3…C_k) ←Decompose(prefix x)
4:     parent←0;
5:     port← -1
6:     for i←1 to k do
7:       length←strlen(C_i)
8:         if(ptr←Lookup(hashtable_i[length], C_i))!=NULL
9:             if(parent ∉ ptr.parent)
10:                break
11:             return -1;
12:             else parent←ptr.code
13:               if ptr.port>0
14:                 port ← ptr.port;
15:               end if
16             end if
17:        else
18:                return -1;
19         end if
20:   end for
21   return port
22: end procedure
```

2.5. Update Mechanism

Update mainly contain two categories: insert and delete prefix. Compared to initialization, inserting needs to extra operation that lookup need to execute in the management list to check if there has valid code. This insertion process is illustrated by

Algorithm 2, where lines 8~14 are used to illustrate update when component exist in the hash table, lines 16~26 inset item into hash table.

Algorithm 2: Insert item into router table

```
1: procedure Insert-Entry(prefix x, port)
2:  k←sumofcomponent(prefix x);//calculate the number of component
3:  (C_1,C_2,C_3…C_k) ←Decompose(prefix x);
4:  parent←0;
5:  for i←1 to k do
6:      length←strlen(C_i)
7:      if ptr←Lookup(hashtable_i[length], C_i) then  //
8:          if(parent ∉ Ptr.parent)
9:            add parent into ptr.parent
10:           end if
11:           if(i==k)
12:           ptr.port←port;
13:           end if
14:            parent← ptr.status;
15:       else if(ptr==NULL)
16:            ptr←allocate_FNLCST_node()
17:            ptr.name← C_i
18:              recover-code←lookup(list_i)
19:                  if(recover-code >0)
20:                    ptr.status← recover-code
21:                    recover-code←0
22:                   else
23:                  ptr.status←status_i
24:                  status_i← status_i+1
25:                   end if
26:            parent←ptr.status
27:          end if
28:      end if
29:  end for
30: end procedure
```

Deleting prefix is relatively complex, reference count is designed to denote the number of times which component are used. This problem will be discussed on three cases: i) If the reference count is bigger than one and the parent node has been not deleted, the reference count only need to decrease one. ii) If the reference count is bigger than one and the parent node has been deleted, the reference count need to decrease one and the parent code should be removed from parent list; iii) If the reference count is equal to one, the hash node should be removed from hash table. This deletion process is illustrated by

Algorithm 3

Algorithm 3: Delete item in router table

```
1: procedure Delete-Entry(prefix x)
2:  k←sumofcomponent(prefix x);//calculate the number of component
3:  (C_1,C_2,C_3…C_k) ←Decompose(prefix x);
4:  parent←0;
```

```
5:  for i←1 to k do
6:      length←strlen($C_i$)
7:      if ptr←Lookup(hashtablei[length], $C_i$) then
8:          if(ptr.ref>1)
9:              ptr.ref← ptr.ref-1
10:             parent← ptr.status
11:             if(flag==true)
12:                 remove the parent from the parent of ptr
13:                 flag←false
14:             end if
15:         else if(ptr.ref==1)
16:             remove the ptr from the $hashtable_i$[length]
17:             add the ptr.status into $list_i$
18:             flag←true
19:         end if
20:     else break
21   end for
22: end procedure
```

3. Time Complexity Analysis

For convenience, we summarize the main notions used in this section in Table 1.

Table 1

symbol	the meaning of symbol
n	the average number of character in the prefix
k	the average number of component in the prefix
w_s	the average length of component
m_c	the average number of child node in the Component Trie
m_t	the average number of child node in the Character Trie
m_s	the average number status which a component belongs to
α	load factor in hash table

In component Trie, the average time complexity matching one component is $o(\mathrm{m}_c \mathrm{w}_s)$, so average time complexity for lookup is $o(\mathrm{km}_c \mathrm{w}_s)$. In the Character Trie, the average time complexity is $o(\mathrm{nm}_t)$ .The length of successful lookup and unsuccessful lookup in hash table can be respectively represented as $1+\frac{\alpha}{2}$ and $\alpha + e^{-\alpha}$, so the average time complexity of FNLCST is $\mathrm{o}(\mathrm{kw}_s \mathrm{m}_s ((1+\frac{\alpha}{2}) + (\alpha + \mathrm{e}^{-\alpha}))$, which gain constant multiple speedup for longest prefix lookup[5].

4. Simulation Results and Analysis

In this section, we evaluate the performance of FNLCST and compare it with other name lookup mechanisms in terms of lookup throughput, memory consumption, update. Character Trie and component Trie [6]are used as contrast algorithm.

4.1. Experimental Setup

The name lookup engine is implemented on Linux operating system with the version Ubuntu13.10. The engine is developed by C program language. Relevant hardware configuration is listed in Table 2.

Table 2. Hardware Configuration

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

4.2 Prefixes and Name

A. 3M Prefix Table

The prefixes table using in experiment, “3M prefix table”, contains 2,739,587 entries. The 3M prefix table is mainly obtained by following steps:

Step 1. The domain name and URL widely used in academic community are downloaded from DMOZ[10], 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 domain names in hash table are extracted to construct prefixes 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 CCN naming convention. For example, ‘www.sina.com’ is modified as ‘ /com/sina/www’.

B. Name trace

The name traces are used to test the lookup performance of our proposed methods[9]. Two types of name traces, average workload 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 with average workload, the heavy workload will consume much more computation.

4.3. The Reuse Ratio of Component in Each Level and Average Reuse Ratio

Reuse ratio in the layer and the average reuse ratio respectively is defined as $r_i = 1 - s_i / c_i$, $r = 1 - s / c$. s_i and c_i represent the number of codes and components in the i$_{th}$ layer respectively, s and c represent the number of codes and components in the whole virtual component Trie respectively[11]. The changing trend is shown as Figure 4.

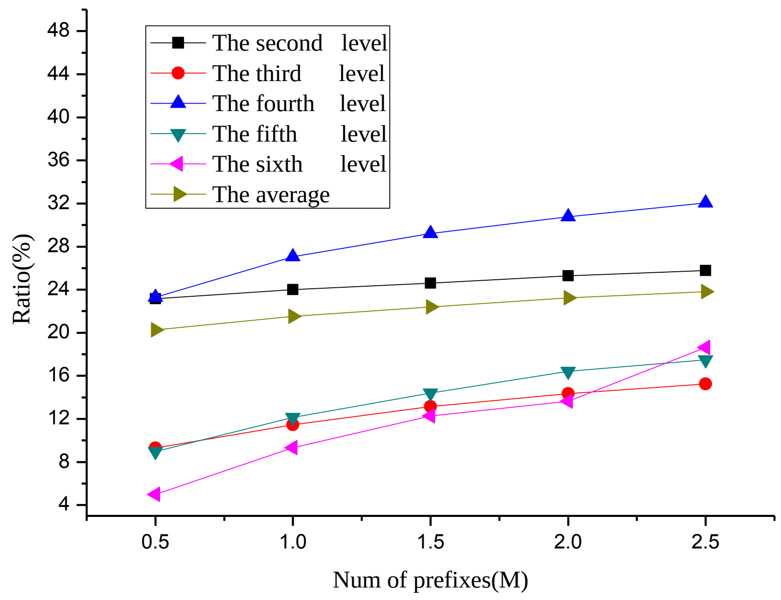

Figure 4. Reuse Ratio For Component

Obviously, the reuse ratio is increasing gradually with the number of prefixes. The average reuse ratio can achieve twenty percent above, which indicate that the number of codes can be saved and memory consumption can be reduced by 20% .

4.4. Memory

The memory main has two parts: 1) hash memory; 2)node memory. The memory comparison with Character Trie, Component Trie are shown in Figure 5.

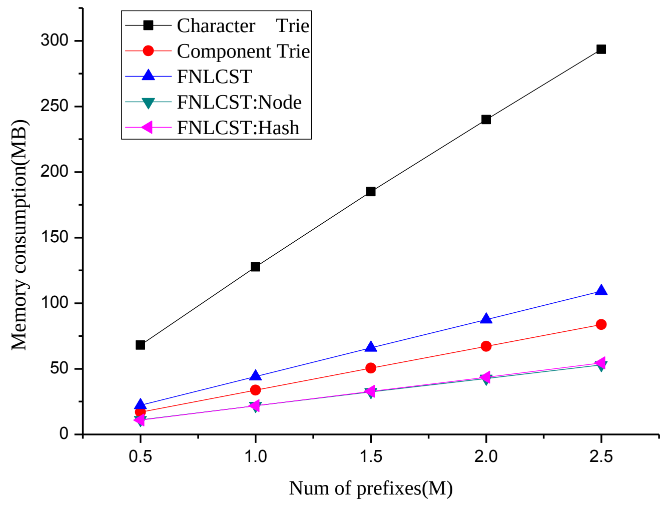

Figure 5. Memory Consumption

From Figure 5, Character Trie consumes the most memory because node only store one character to result too many nodes. In order to update better, reference count and hash are introduced for time with space, so the memory cost need a little more than component Trie.

4.5. Lookup Speed

The name trace that used to simulate the name of Interest Packet are imported to test the lookup performance in different router scale under average workload and heavy workload. The comparison with Character Trie, Component Trie are shown in Figure 6.

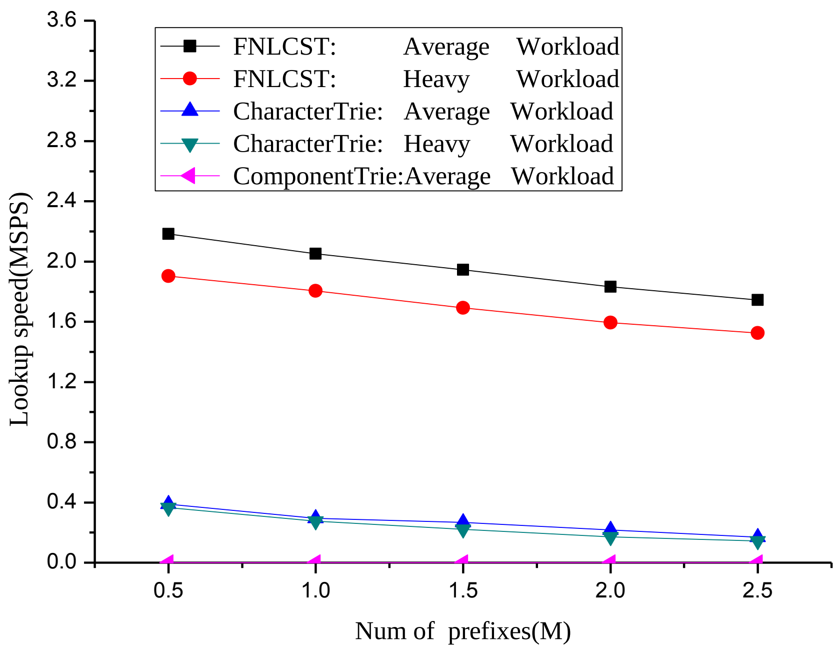

Figure 6. Lookup Speed

Lookup speed decline slightly with the size of router increasing. The performance of FNLCST are enhanced greatly compared to other two algorithms. Because of the special nature of Character Trie each node has variety children and node can only store one character to lead deep depth, so the performance is bad. Comparing with character Trie, the depth of component Trie can be degraded. But the degree of polymerization drop because of bigger granularity, so the node has too many children which bring too much comparison in one component matching. In FNLCST, the virtual component Trie has the same depth compare with component Trie. In addition, components that grouped by the length can disperse their children. And the number of memory access times can be reduced because of no character pointer.

4.6. Update Speed

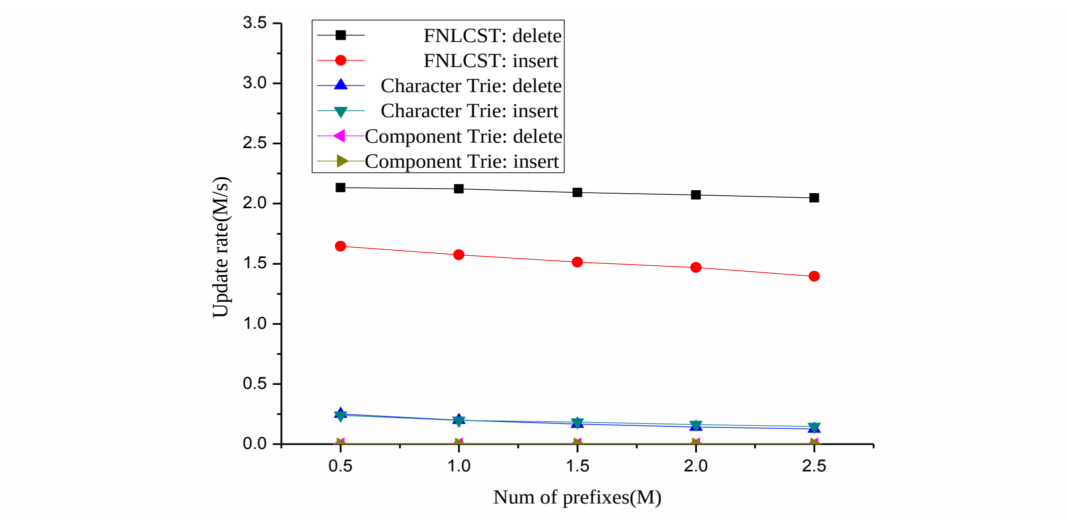

Figure 7. Update Speed

Twenty percent of the items are treated as input to test the update performance at different router scale. The deleted prefixes should be contained in the router table, but the inserted prefixes shouldn't be contained to guarantee update performance accurately. The comparison with Character Trie, Component Trie are shown in Figure 7.

The performance of FNLCST is better than other two algorithms, because hash function and reference count can speedup update. In character Trie, the depth is too deep to result many memory accesses. The node has great number of children because granularity is increasing, so the performance degrade greatly in component Trie.

5. Conclusion

Hash function, component encoding and Component Trie are combined to overcome the shortcoming of Component Trie. At the same time, we propose that components in the level in the Component Trie are grouped by length to speedup lookup. Simulation results show that FNLCST has better lookup speed and update performance. As part of our future work, we will make deep research on hash function to speedup name lookup.

Acknowledgments

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] B. Ahlgren, C. Dannewitz and C. Imbrenda, "A Survey of Information-Centric Networking (Draft)", Communications Magazine IEEE, vol. 50, no. 7, **(2011)**, pp. 26 - 36.
[2] L. Zhang, D. Estrin and J. Burke, "Named data networking(NDN) Project", Technical Report, NDN-0001, **(2010)**. [Online].Available: http://www.named-data.net/.
[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs and R. L. Braynard, "Networking Named Content", In CoNEXT09, **(2009)**, pp. 1-12
[4] M. Bienkowski and S. Schmid, "Competitive FIB aggregation for independent prefixes: online ski rental on the trie", Revised Selected Papers of International Colloquium on Structural Information & Communication Complexity. Springer-Verlag New York, Inc., **(2013)**, pp. 92-103.

[5] Y. Wang, K. He and H. Dai, "Scalable Name Lookup in NDN Using Effective Name Component Encoding", Proceedings - International Conference on Distributed Computing Systems, **(2012)**, pp. 688-697.
[6] T. Zhang, Y. Wang and T. Yang, "NDNBench: A benchmark for Named Data Networking lookup", GLOBECOM 2013 - 2013 IEEE Global Communications Conference, **(2013)**, pp. 2152-2157.
[7] 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, vol. 10, no. 1109, **(2013)**, pp. 95-99.
[8] H. Dai, J. Lu and Y. Wang, "BFAST: Unified and scalable index for NDN forwarding architecture", Computer Communications (INFOCOM), 2015 IEEE Conference on. IEEE, **(2015)**.
[9] W. So, A. Narayanan and D. Oran, "Toward fast NDN software forwarding lookup engine based on hash tables", Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems, **(2012)**, pp. 85-86.
[10] DMOZ-open directory project [EB/OL]. [2**011**-03-30] http://www.dmoz.org/.
[11] W. Quan, C. Xu and J. Guan, "Scalable Name Lookup with Adaptive Prefix Bloom Filter for Named Data Networking", IEEE Communications Letters, vol. 18, no. 1, **(2014)**, pp. 102-105.

Authors

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.

Jianghua Xu, he has got master of engineering degree from Chongqing University of posts and telecommunications. His main research interests include Name Lookup in Name Data Networking, IP Lookup and Software Development.

Sheng Huang, he 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 include optical communication system, channel coding and networks.

Sijun Liu, he is a postgraduate student of Chongqing university of posts and telecommunications.His main research interests is caching in Content Center Network.