메모리 제약을 가진 LC-Trie를 이용한 IP 참조 테이블 디자인

이채영[1] · 박재규[2]
한국과학기술원 산업공학과

IP Lookup Table Design using LC-trie with Memory Constraint

Chae Y. Lee[1] · Jae G. Park[2]
Department of Industrial Engineering, KAIST

IP address lookup is to determine the next hop destination o f an incoming packet in the router. The address lookup is a major bottleneck in high performance router due to the increased routing table sizes, increased traffic, higher speed links, and the migration to 128 bits IPv6 addresses. IP lookup time is dependent on data structure of lookup table and search scheme.
In this paper, we propose a new approach to build a lookup table that satisfies the memory constraint. The design of lookup table is formulated as an optimization problem. The objective is to minimize average depth from the root node for lookup. We assume that the frequencies with which prefixes are accessed are known and the data structure is level compressed trie with branching factor *k* at the root and binary at all other nodes. Thus, the problem is to determine the branching factor *k* at the root node such that the average depth is minimized. A heuristic procedure is proposed to solve the problem. Experimental results show that the lookup table based on the proposed heuristic has better average and the worst-case depth for lookup.

key words: IP, lookup table, LC-Trie, memory constraint

1. Introduction

Internet consists of different sub-networks which are interconnected via gateways, routers, switches, and various transmission facilities. IP (Internet Protocol) provides the functionality for connecting end systems across multiple networks which may use different protocols. Today's Internet provides a broad range of services from E-mail to complex client/sever applications such as the World Wide Web (WWW), Video On Demand (VOD) service and e-commerce. Statistics show that the number of hosts on the Internet is tripling approximately every two years and traffic of Internet increases exponentially (Lampson, B. *et al*., 1998)

[1] Department of Industrial Engineering, KAIST, 373-1 Kusung Dong, Yusung Gu, Taejon, 305-701, Korea
Fax: +82 42 869 3110
E-mail: cylee@mail.kaist.ac.kr
[2] Department of Industrial Engineering, KAIST, 373-1 Kusung Dong, Yusung Gu, Taejon, 305-701, Korea
E-mail: uta@kaist.ac.kr

Speeding up the packet forwarding in the Internet backbone requires both high-speed communication links and high performance router. While advances in optical technologies such as WDM (Wavelength Division Multiplexing) have increased link speeds to beyond tens of gigabits per second, Internet backbone routers have been lagging behind. One main reason for this is the relatively complex packet processing required at a router. For every incoming packet, the router has to lookup a packet's next hop destination. As a result, the major bottleneck remained in the Internet router is slow software-based IP lookup procedures. Especially, because the lookup speed at the router is dependent on the data structure used for the lookup table, the design of the table for high speed lookup has received considerable attention in telecommunication research.

When IP was first standardized in 1981, the designers decided that the IP address space should be divided into three different address classes- Class A, Class B, and Class C (Comer, D.E., 1995). This is often referred to as "classful" addressing because the address space is split into predefined categories. Each class fixes the boundary between network part and host part within 32 bits. This simplified the routing system because the classes are identified with first 3 bits. But the exponential growth of the Internet raised serious concerns about exhaustion of IP address. To solve this address problem, the concept of CIDR (Classless Inter-Domain Routing) was developed (Fuller, V. *et al.*, 1993) which eliminates the traditional concepts of classes and replaces them with the generalized concept of "network prefix". Also IPv6 is currently being explored in a working group of IETF. IPv6 uses 128 bits instead of 32 bits in the IPv4. The network prefix enables the efficient allocation of the IPv4 address space until IPv6 is deployed.

In the way of CIDR, each IP route is identified by a <route prefix, prefix length> pair, where the prefix length is in the range between 0 and 32 bits. For every incoming packet, a search in the router's lookup table must be performed to determine which next hop the packet is destined for (Fuller, V. *et al.*, 1993). With CIDR, the search may be decomposed into two steps. First, we need to find the set of routes whose prefixes match that of the incoming IP destination address. Then, among the set of routes, we have to select the one with the specifically matching prefix. This is the route used to identify the next hop. Due to the facts that table entries have variable lengths and that multiple entries may represent the valid routes to the same destination, the search may be complicated. This operation is called the "longest prefix matching" (LPM). The LPM operation is the most important process of IP lookup because an invalid route cannot give correct destination information. In the primary addresses of Class A, Class B, and Class C, we can use exact matching prefix because the network parts are only 8, 16, and 24 bits respectively for addressing (Comer, D. E., 1995). But, in the CIDR, we cannot use exact matching prefix for lookup procedure because of variable prefix lengths. Thus, we cannot use the traditional binary tree data structures which only process exact matching procedure.

We present a general framework for generating optimal lookup table for IP routing with memory constraint. For the longest prefix matching we use level compression trie data structure, which is known to support the LPM. We formulated the IP lookup problem as an optimization problem, and suggested heuristic approach for building the lookup table.

In Section 2, we examined the existing approaches for IP lookup. In Section 3, a heuristic approach for building lookup table is presented. Simulation results based on the proposed approach are discussed in Section 4. Finally, a conclusion is given in Section 5.

2. Existing Approaches to IP Lookup

There have been many schemes developed to solve the classless IP lookup problem. Some schemes are hardware specific, while others are more suitable for software solutions, and some others take protocol based approaches.

In hardware specific approaches, they use hashing functions for indexes into memory and cash memory. In the worst as well as average cases, with proper design of such functions, significant improvements in lookup speeds can be obtained. There are approaches using CAM (McAulley, A. *et al.*, 1993), Caching (Gupta, P. *et al.*, 1998), and Large Memory Architecture in this area.

In software-based approaches, they use data structures such as a modified binary tree (it is also called binary trie), Patricia tree, and Level Compression Trie for the IP lookup (Cheung, G. *et al.*, 1999).

In case of using binary tree, each prefix is represented by a node in the tree, usually a leaf, and the path from the root to that node reveals the associated bit pattern of the prefix. Each node of the tree has at most two children, hence is called binary tree, and it is one case of trie which may have more than two children. In this approach, the lookup performance is not so good because the worst case of memory access for lookup is 32 times which is the longest bit length of IPv4. Patricia tree is a tree where certain sequences of input bits may be skipped over before arriving at the next node. Compression is occurred when a long sequence of one nodes needs to be traversed before arriving at a subtree of prefixes. We can reduce the lookup times by skipping compressed nodes. As a variation of the Patricia tree, level compression trie is introduced for efficient lookup operation (Nilsson, S. *et al.*, 1999).

The level compression is a trie in which each complete subtree of height *h* is collapsed into a subtree of height one with 2^h children (Nilsson, S. *et al.*, 1999). For example, starting at node *i*, if there are exactly 2^h descendent nodes at level *h*, then all the intermediate nodes from level 1 to level *h* –1 can be replaced by 2^h branches originated directly from node *i*. With the level compression trie, we only need one comparison at the binary tree instead of *h* comparisons. However, the existing level compression trie is only built in a complete tree case; the data structure does not consider memory constraint. In this paper, we suggest modified level compression trie which satisfies the memory constraint while minimizing average depth from

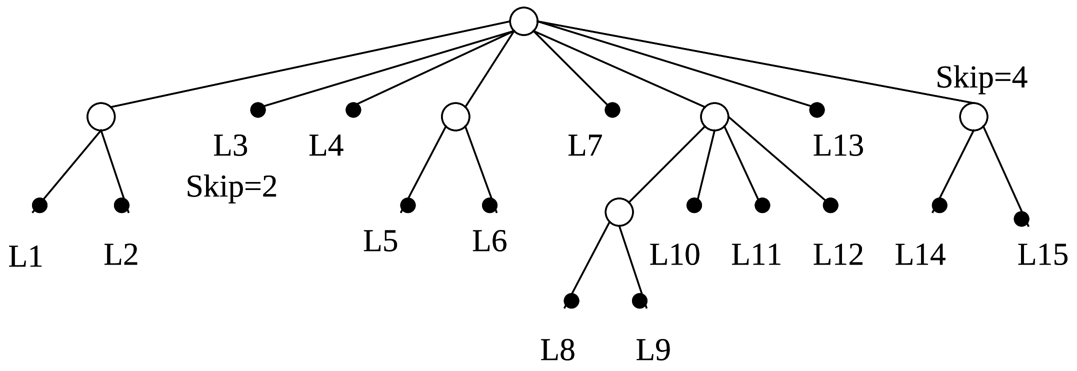

Figure 1. LC Trie representation

the root node.

Protocols based approaches are the ways to have extra information sent along with the packet to simplify or even totally get rid of IP lookups at routers. Two major proposals along these lines are IP Switching and Tag Switching (Rechter, Y. *et al*., 1997). However both schemes require large network modification to adopt their schemes for lookup operation. As an example, ingress router is required to perform a full routing decision.

3. Heuristic Approach for Building Lookup Table

Although previous works on the fast IP route lookup proposed elegant designs for their table lookup schemes, no approaches attempted to design an optimal strategy except some papers (Cheung, G. *et al*., 1999, Gupta, P. *et al*. 2000). In this paper a lookup table which considers memory constraint for the branching factor, skip value and pointer is considered that minimizes average depth of the branches in the table. The depth represents the number of memory access to get the next hop information of each prefix.

We first present a brief introduction to the data structure used in IP address lookup and representation of how prefixes are mapped to its structure. We then present a formulation of the lookup table design problem to determine the branching factor at the root node. Finally we propose a heuristic procedure to design an IP lookup table which is modified from level compression trie.

Table 1. Binary strings to be stored
in a trie structure

Prefix	Next hop
0 0 0 0	L1
0 0 0 1	L2
0 0 1 0 1	L3
0 1 0	L4
0 1 1 0	L5
0 1 1 1	L6
1 0 0	L7
1 0 1 0 0 0	L8
1 0 1 0 0 1	L9
1 0 1 0 1	L10
1 0 1 1 0	L11
1 0 1 1 1	L12
1 1 0	L13
1 1 1 0 1 0 0 0	L14
1 1 1 0 1 0 0 1	L15

3.1 Representation of Level Compression Trie

Consider an example of binary strings given in Table 1. A string 010 corresponds to the path from the root to the node with next hop information, L4. The prefixes in the table can be represented to level compression trie shown in Figure 1. Table 2 shows the array representation of the LC trie in Figure 1 in the memory. Each entry represents a node.

Nodes in Table 2 are numbered in breadth-first order starting at the root in Figure 1. Each number k in the branching factor indicates the number of bits used for branching at the corresponding node. A value $k \geq 1$ indicates that the node has 2^k children. The value $k = 0$ means that the node is a leaf. The skip value represents the number of bits that can be skipped during the search operation. When there is a long sequence of one child in the trie structure, the memory space as well as the depth for searching the node can be reduced by using the skip value. Comparisons for the prefix bits corresponding to the skip value are not

Table 2. Memory structure of LC trie

Node	Branching Factor (k)	Skip Value	Pointer of the leftmost child	Corresponding Next hop: Prefix
1	3	0	2	
2	1	0	10	
3	0	2	0	L3 : 0 0 1 0 1
4	0	0	0	L4 : 0 1 0
5	1	0	12	
6	0	0	0	L7 : 1 0 0
7	2	0	14	
8	0	0	0	L13:1 1 0
9	1	4	18	
10	0	0	0	L1 : 0 0 0 0
11	0	0	0	L2 : 0 0 0 1
12	0	0	0	L5 : 0 1 1 0
13	0	0	0	L6 : 0 1 1 1
14	1	0	20	
15	0	0	0	L10:1 0 1 0 1
16	0	0	0	L11:1 0 1 1 0
17	0	0	0	L12: 1 0 1 1 1
18	0	0	0	L14: 1 1 1 0 1 0 0 0
19	0	0	0	L15: 1 1 1 0 1 0 0 1
20	0	0	0	L 8 : 1 0 1 0 0 0
21	0	0	0	L 9: 1 0 1 0 0 1

needed because there is only one path. Memory reduction is occurred by compression of the nodes for the skip value.

The value in the pointer column represents a pointer to the leftmost child node. The pointer value, zero represents the corresponding node is a leaf node which has the next hop information.

As an example, suppose we search for the prefix 11101001. We start at the root node, node 1, represented in the first column. Since the branching factor *k*=3 we extract the first three bits from the search prefix. Note that the root node has 2^3=8 children nodes from 000 to 111. Since the pointer of the leftmost child of the root node is 2 (000) we access node 9 (111). At node 9 the value of branching factor is 1, skip value is 4, and the pointer value is 18. Because the skip value is 4, we skip the next 4 bits, 0100. Then we access the node 19 the last bit is 1. At node 19 we finally get the next hop information, L15, since it is a leaf node with the branching factor *k*=0.

3.2 Formulation

Note that today's routers already maintain per-prefix statistics. Thus our question is " What is the best trie data structure given the frequency of access of prefix with memory constraint? " Viewing it this way, the problem is readily recognized to be one of minimizing the average weighted length of a trie problem whose prefixes are weighted by the access probabilities. In other words, the problem is to determine the branching factor at each node with memory constraint. However, since the decision of branching factor at each node in each level vastly increases the problem complexity, we are interested in the branching factor at the root node. The branching at all descendant nodes are assumed binary. In this section, we formulate the problem as a minimization of average depth of the prefixes.

To solve the minimization problem of the average weighted length with memory constraint, we present the following basic notation. We assume that there are N prefixes each of which has access frequencies, p_i, and length of L_i with memory constraint *M*. Let the branching factor at the root node is given by *k*, which is a decision variable. Then the depth of $i\text{-}th$ prefix, $d_i(k) = L_i - k + 1$ when the prefix length is greater than branching factor. Note that the first *k* bits are used for the branching information at the root node, and $L_i - k$ bits are for the branching at all descendant nodes. Since the branching at each descendant node is binary, the total depth becomes $L_i - k + 1$. Otherwise, $d_i(k)$=1.

Let MU be the memory space for storing information such as branching factor, skip value, and leftmost children's pointer at each node. Let M_k be the sum of required memory for nodes at the 2^k sub-binary trees branched from the root node with branching factor *k*. Then the maximum positive integer k is determined by $T = \lfloor \log_2\{(M - MU)/MU\} \rfloor$, which occurs when all nodes are directly branched from the root node.

N : total number of prefix
k : branching factor of root node
p_i : access frequency to prefix i

$d_i(k)$: depth of prefix i
M : memory size which is a constraint
MU : memory unit required for storing information of one node
M_k : sum of memory size for sub binary trees with branching factor *k* at the root node
L_i : length of prefix i

Now, the minimization problem of the average weighted length with memory constraint is formulated as follows.

$$\text{Minimize} \quad \sum_{i=1}^{N} p_i d_i(k) \qquad (1)$$

Subject to

$$(2^k + 1)MU + M_k \le M \quad \text{For } k = 1,2,\dots,T \qquad (2)$$

$$d_i(k) = \begin{cases} L_i - k + 1 & \text{if} \quad L_i - k \ge 0 \\ 1 & \text{if} \quad L_i - k < 0 \end{cases} \qquad (3)$$

$$\text{For } i = 1,2,\dots,N$$

The objective (1) is to determine the branching factor k at the root node to minimize the sum of weighted depths of prefixes. Constraint (2) shows the memory constraint when the root node has a branching factor *k*. Since the root node has 2^k children and each node requires 1*MU* for storing node information, the memory requirement for the information at the root node and 2^k children becomes $(2^k + 1)MU$. M_k is for all other nodes at the 2^k sub-binary trees. Constraint (3) is related to the depth of each prefix when branching factor as explained in this section.

Here we will briefly discuss the complexity of the above problem. Note that most of prefix lengths L_i are between 16~24 bits (Nilsson, S. *et al.*, 1999). When we restrict the depth $d_i(k)$ of a lookup table to 4~12, the branching factor, *k* becomes 4~20. Now our objective is to determine which prefix or part of prefix bits to select among 1,000s or 10,000s prefixes to form the *k* branches. Clearly, the problem cannot be handled with any traditional mathematical programming procedure. We thus present a heuristic procedure to solve the weighted minimum depth problem with memory constraint.

3.3 Heuristic Procedure for building lookup table

In this section, we propose a heuristic procedure for building a lookup table based on level compression trie. The procedure consists of three steps. The first step is to make an initial solution that transforms prefixes to binary tree. In the second step, a modified level compression trie is built. Finally in the third step, the modified level compression trie that satisfies the memory constraint is considered.

In the second step, the procedure repeatedly searches a node that can reduce the required memory by collapsing its children nodes. We define this node as a decrement node. Note that the existing method (Nilsson, S. *et al.*, 1999) reduces memory by collapsing only a complete binary tree. In other words, compression is performed only when a node has a complete binary tree. In the proposed procedure, however, compression is processed when there is a decrement node even if it is not a complete binary tree. Thus, the proposed procedure reduces the average depth of the lookup table.

The third step is to build the modified level compression trie that satisfies the memory constraint. In this step, the procedure increases branching factors of nodes within memory constraint by taking access frequencies into account. A node which has maximum sum of probabilities of their grandsons is chosen as a new branch. This operation also has the effect of reducing average depth. The heuristic procedure is terminated when there is no node which increases the branching factor within memory constraint.

Proposed Heuristic Procedure

```
BEGIN

Step 1: Initial solution
1. Sort prefixes in increasing order
2. Make an initial solution with binary tree structure
3. Go to Step 2

Step 2: Building the LC-trie
1. Compute the memory decrements of each node when its children is compressed
        IF there is no decrement nodes
            THEN go to Step 3
2. Choose the node which has a maximum decrement and compress its children
        IF there exists same maximum decrement
            THEN choose the node of lower level
3. Build the new trie
4. Compute the decrement of memory when each level of new trie is compressed
            IF there is no memory decrement of nodes greater than or equal to zero
              THEN go to Step 3
5. Go to Step2-2

Step 3: Building the LC-trie which satisfies the Memory constraint
1. Check whether LC-trie satisfies the Memory constraint
          IF it does not satisfy the Memory constraint
              THEN stop and return the infeasible message
```

```
2. Choose the node which has maximum sum of probabilities of its grandsons
        IF the required memory for compression of its children nodes does not satisfy the memory
           constraint
            THEN return as final solution
3. Build the new trie
4. Compute the required memory for compression of its children when each node of new trie is compressed
        IF there is no level which satisfy the memory constraint
            THEN stop and return as final solution
5. Go to Step 3-3

END
```

4. Simulation Results

Statistics (Nilsson, S. *et al.*, 1999) shows that most of prefix lengths are between 16 bits and 24 bits, especially the prefixes whose lengths are 24 bits are more than half of total prefixes. Thus we randomly generated 50 % of total prefixes for prefixes of 24 bits length, and 30 % for prefixes between 16bits and 23 bits. Other prefixes are generated with the rest of portion. We also gave access frequencies for each prefixes by random generation such that the sum of the access frequencies is equal to one.

Two experiments are performed to investigate the performance of the proposed heuristic. The first experiment is performed without memory constraint. Table 3 shows the simulation result. Ten sets of instances are experimented in each case with different number of prefixes. The proposed heuristic is compared to one of existing approaches which is built in the complete tree case (Nilsson, S. *et al.*, 1999). The required memory in the table represents the number of entries for storing the prefixes. Each number under the required memory is the smallest integer that is greater than the averaged memory size. Clearly, the proposed heuristic has better performance in all cases with reduced memory. The average and worst case depth are dramatically reduced. The result also shows that the average depth becomes smaller as the number of prefixes increases. This can be explained by the fact that the level compression is more efficient in a lookup table with more densely populated prefixes.

Another experiment is performed with different memory constraint. For each size of prefixes ten sets of instances are generated each with different memory constraint. As shown in Table 4, the heuristic gives lower average depth as the memory constraint increases. The depth in the worst case is also reduced with the proposed heuristic, which illustrates an efficient level compression trie for IP lookup table.

Table 3. Comparison of the Proposed and Existing Procedures without Memory Constraint

Number of Prefixes	Average Depth		Depth in the Worst Case		Required Memory	
	Proposed Heuristic	Existing Algorithm	Proposed Heuristic	Existing Algorithm	Proposed Heuristic	Existing Algorithm
100	6.72	10.92	12	15	131	167
1000	7.03	11.25	11	21	1220	1591
5000	5.83	9.83	13	17	6290	7993
10000	5.47	8.82	9	14	12685	15873
15000	4.42	6.08	8	15	18018	23833

Table 4. Performance of Proposed Heuristic with different Memory Constraints

Memory Constraint	5000 Prefixes		10000 Prefixes		15000 Prefixes	
	Average Depth	Depth in the Worst Case	Average Depth	Depth in the Worst Case	Average Depth	Depth in the Worst Case
20000	3.43	7	3.66	7	4.21	8
30000	2.91	6	3.02	7	3.68	7
40000	2.58	5	2.76	5	2.88	6
50000	2.09	4	2.31	4	2.43	5
60000	1.99	4	2.10	4	2.31	4

5. Conclusion

In this paper, the lookup table design problem is considered to minimize the average depth required to obtain the next hop information. A heuristic procedure is proposed to minimize the average depth with memory constraint. Experimental results show that the improvement is overwhelming compared to one existing algorithm. Lower depth is obtained with less memory compared to the existing procedure. The improvement will be much more visible in case of IPv6 which requires much more memory due to increased number of address bits.

Reference

Cheung, G. and McCanne, S. (1999), Optimal Routing Table Design for IP Address Lookups under Memory Constraint, *Proceedings of INFOCOM'99*.

Comer, D. E. (1995), *Internetworking with TCP/IP*, Volume 1, Third edition, Prentice Hall.

Fuller, V., Li, T., Yu, J., and Varadhan, K. (1993), Classless inter-domain routing (CIDR): An address assignment and aggregation strategy, *RFC 1519*.

Girish, P.C., Varghses, G. (1996), Trading Packet Headers for Packet Processing, *IEEE/ACM Transactions on Networking*.

Gupta, P., Lin, S., McKeown, N. (1998), Routing Lookups in Hardware at Memory Access Speeds, *Proceedings of IEEE INFOCOM'98*, 1240-1247.

Gupta, P., Prabhakar, B., Boyd, S. (2000), Near Optimal Routing Lookups with Bounded Worst Case Performance, *Proceedings of INFOCOM 2000 Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 3, 1184 -1192.

Lampson, B., Srinivasan, V., Varghese, G. (1998), IP Lookups using Multiway and Multicolumn Search, *INFOCOM'98*, 3, 1248-1256.

McAulley, A., Fancis, P. (1993), Fast routing table lookup using CAM's, *Proceedings of INFOCOM'93* , 3, 1382-1391.

Nilsson, S., Karlsson, G. (1999), IP Address Lookup Using LC-Tries, *in Selected Area Communications, IEEE Journal*, 176, 1083 –1092.

Rechter, Y., Davie, B., Katz, D., Rosen, E., and Swallow, G. (1997), Cisco System's Tag Switching Architecture Overview, *Technical Report RFC 2105*.