Memory Efficient IPV4/V6 Lookup using Path Compression

C.Shalini
PG student
Department of CSE
Government College of Engineering, Tirunelveli

D.Shalini Punithavathani
Professor & HOD
Department of CSE
Government College of Engineering, Tirunelveli

ABSTRACT

With the rapid growth of the Internet, IP-lookup becomes the bottle-neck in network traffic management. Therefore, the design of high speed IP routers has been a major area of research. The focus of this paper is on achieving significant reduction in memory requirements for the longest prefix-match operation needed in IPv4/v6 lookups. The Longest Prefix Matching (LPM) is one of the problems in the uni-bit trie representation, in which the number of nodes and the memory requirement is high for IP lookup. To solve this problem we propose a classic trie-based approach in IP lookup. We propose an algorithm to compress the uni-bit-trie representation of a given routing table by using single-prefix distance bounded path compression algorithm. This algorithm determines the optimal maximum skip distance at each node of the trie to minimize the total memory requirement.

Keywords

IP-lookup, longest prefix matching, skip distance, routing tables

1. INTRODUCTION

Memory efficiency with compact data structures for Internet Protocol (IP) lookup has recently gained much interest in the research community. Internet Protocol (IP) lookup in routers can be implemented by some form of tree traversal [2]. The length of the IP address is 32 bits in IPv4 and 128 bits in IPv6 [3]. This increased length allows for a broader range of addressing hierarchies and a much larger number of addressable nodes. Every machine that is on a TCP/IP network (a local network, or the Internet) has a unique Internet Protocol (IP) address.

According to the Internet Architecture Board (IAB), an IPv6 address consists of two parts: a 64-bit network/sub-network ID followed by a 64-bit host ID. At the core router, only the 64-bit network/sub-network ID is relevant in making the forwarding decisions. The increase in the size of routing table and the extension of the prefix length necessitate high memory-efficient lookup algorithms to reduce the size of the storage memory. Existing compression techniques [4], [5] can increase the total memory requirement and the computational complexity at the nodes, while reducing the total number of nodes.

The main objective for this paper is on achieving significant reduction in memory requirements for the longest prefix match operation needed in IPv4/v6 lookups. So the distance-bounded path compression algorithm for trie-based IP lookup is proposed. It makes the single-prefix distance-bounded path compression. This algorithm can compress the trie of a given routing table to significantly reduce the total memory requirement.

The distance-bounded path compression [1] algorithm supports over 330K IPv4/v6 prefixes and sustains a high throughput of 466 million lookups per second. The key issues to be addressed in designing architecture [2] for IP packet forwarding engine are: size of supported routing table, throughput, scalability (with multiple chips or external storage), in-order packet output, incremental update, and power consumption.

2. EXISTING SYSTEM

In trie-based approaches [6], [7], [8], [9] IP- lookup is performed by simply traversing the trie according to the bits in the IP address. These designs have a compact data structure, but large number of trie nodes; hence, moderate memory efficiency.

2.1 IP lookup

IP Lookup is considered the core function of a router. Internet Protocol (IP) lookup in routers can be implemented by some form of tree traversal. The nature of IP lookup is longest prefix matching (LPM). LPM refers to an algorithm used by routers in IP networking to select an entry from a routing table.

2.2 Longest prefix matching

The IP lookup operation requires a longest matching prefix computation at wire speeds. At every hop (router), for each packet of the destination IP address is matched against a database of IP prefixes. Each prefix entry consists of a prefix and a next hop value.

Table 1: A Sample Routing Table (Maximum Prefix Length=8)

	Prefix	Next Hop		Prefix	Next Hop
P_0	*	0	P_5	11111*	5
P_1	0010*	1	P_6	0101*	6
P_2	01100*	2	P_7	1*	7
P_3	010111*	3	P_8	0*	8
P_4	110101*	4			

In this routing table [10], binary prefix P1 (0010∗) matches all destination addresses that begin with 0010. Similarly, prefix P4 matches all destination addresses that begin with 110101. The 8-bit destination address IP= 01011111 is matched by the prefixes P0, P3, P6, and P8. Since |P0| = 0, |P3| = 6, |P6| = 4, |P8| = 1. Here P3 is the longest prefix that matches IP (|P| is defined as the length of prefix P). In longest prefix routing, the next hop index for a packet is given by the table entry corresponding to the longest prefix that matches its destination IP address.

For an incoming packet, its destination address is compared with all the current prefixes in the routing table and the next-hop associated with the longest matching prefix [11] is determined to be the output port for the packet.

2.3 Trie-based IP Lookup

The most common data structure in algorithmic solutions for performing LPM is some form of trie [12]. Trie is a binary tree, where a prefix is represented by a node. The value of the prefix corresponds to the path from the root of the tree to the node representing the prefix [13]. The branching decisions are made based on the consecutive bits in the prefix.

A trie is called a uni-bit trie if only one bit is used for making branching decision at a time. The routing table can be drawn as a binary search tree is called uni-bit trie. The uni-bit trie can be compressed as in [14], [4], [5] to reduce the number of nodes. The compression techniques can greatly increase the total memory requirement and the computational complexity at the nodes.

The prefix set in table 1 corresponds to the uni-bit trie. The prefix bits are scanned from left to right. If the scanned bit is 0, the node has a child to the left and a bit of 1 indicates a child to the right. Normally, each trie node contains two fields: the represented prefix and the pointer to the child nodes.

The uni-bit trie, IP lookup is performed by traversing the trie according to the bits in the IP address. When a leaf is reached, the prefix associated with the leaf is the longest matched prefix for that IP address. The time to look up a uni-bit trie (which is traversal in a bit-by-bit fashion), is equal to the prefix length. The use of multiple bits in one scans increase the search speed. Such a trie is called multiple bit trie. The number of bits scanned at a time is called stride.

Existing designs cannot support large IPv6 routing tables consisting of over 300k prefixes. The memory requirement also increases for routing tables with high percentage of distinct prefixes. It is also unclear as how to scale these designs to support larger routing tables and/or longer prefix lengths.

3. PROPOSED SYSTEM

Path compression techniques work well to reduce the number of trie nodes [14]. Yet, reducing the total number of nodes does not necessarily lead to the reduction of the memory footprint. The ultimate goal is to reduce the total required storage.

This paper focuses on the classic trie-based approach for IP lookup. To overcome the limitations of the LPM, we proposed the single-prefix distance-bounded path compression algorithm.

3.1 Single-Prefix Distance-Bounded Path Compression (SPDBPC):

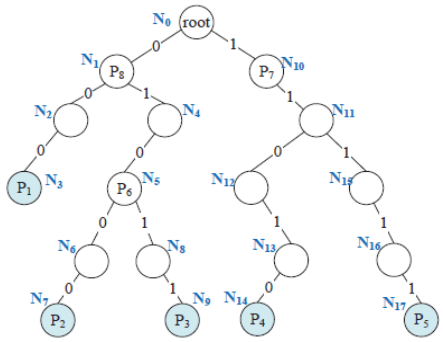

Fig 1: Sample Trie

This is the sample trie for the single-prefix distance-bounded path compression algorithm. By using this, we have to perform the compression for this trie and also find skip distance. In this path compression algorithm the length of a non-branching path in a trie varies from 1 to the maximum depth of the trie.

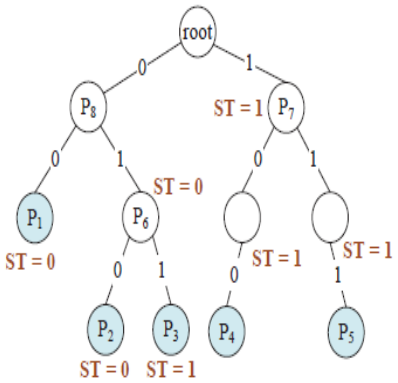

Fig 2: Compressed trie for skip-distance=1(ST=skip string)

This compressed trie is designed from the sample trie with the maximum skip-distance of 1, and the total number of nodes is 11. A memory-efficient implementation of path compressed trie requires: (1) the skip distance to be bounded and (2) the optimal maximum skip-distance *D* to be determined to minimize the memory requirement.

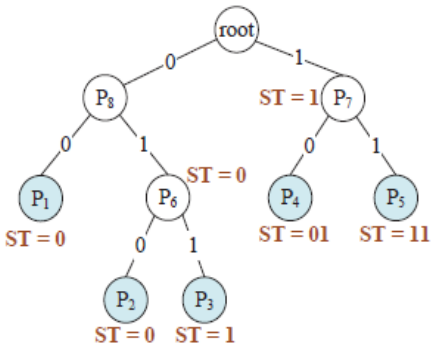

Fig 3: Compressed trie for skip-distance=2

The above compressed trie is designed from the figure 2; with the total number of nodes are 9, and the maximum skip-distance of 2.

Each node of the SP-DBPC trie has the following data fields: (1) 2 child pointers, (2) skip-string, (3) length of actual skip-string, (4) position of prefix bit, and (5) next hop information.

3.2 Algorithm 1: SP-DBPC

Input: Current trie node *n*, maximum skip-distance *D*

Output: Compressed trie

Initialize *m* = # of trie nodes to be merged and *d* = skip-distance

Find the non-branching path *P* of the current node, and calculate *d*

Check whether $d \leq D$ then

Set $m = d$

Otherwise

Set $m = D$

Merge *m* nodes of *P* to the current node

Update the skip-string of the current node

Update children of the *super-node*

SP-DBPC ($n \rightarrow$ *left child*, *D*)

SP-DBPC ($n \rightarrow$ *right child*, *D*)

3.3 Procedure for SP-DBPC

The single-prefix distance-bounded path compression algorithm works as follows.

- First, *SP-DBPC* finds the non-branching path *P* at each node of the trie (starting from the root), and then calculates the skip-distance *d* for the current node.
- Let *m* denote the number of nodes following the current node on path *P* that can be merged with the current node. The number of nodes (m) can be calculated as min (*d*, *D*).
- After the calculation is done the skip-string of the current node is updated. Now the child-nodes of the last merged node become the child-nodes of the current node.
- Finally, The SP-DBPC algorithm is then executed recursively until the leaf-node is reached. Once the algorithm completes, the total number of nodes is calculated and returned.

3.4 Search in a SP-DBPC trie

Trie search algorithm is a good choice for IP forwarding engine due to its simple search at each node. However, in a sparse trie such as one found in IPv6, the number of nodes in the trie drastically expands as the prefix length increases from 32 bits to 64 bits. After building the compressed trie, IP-lookup operation can be performed as follows.

The destination IP address is extracted from the incoming packet. At each node of the compressed trie, there are 3 steps to be executed. The steps are,

1) The skip-string and its skip-distance *d* are extracted. If $d = 0$, skip to Step 3.

2) The skip-string is compared with the next *d* bits of the IP address. If there is no match and the current node is not a prefix-node, then the search is terminated. Otherwise, the next hop information is updated and the search is terminated.

3) If the current node is a prefix-node, then the next hop information is updated and the IP address is left-shifted by (d+1) positions. If a leaf-node is reached, then the search is terminated; otherwise, go back to Step 1.

4. CONCLUSION AND FUTUREWORK

This project is on achieving significant reduction in memory requirements for the longest prefix match operation needed in IPv4/v6 lookups. This proposed scheme determines the optimal maximum skip distance at each node of the trie to minimize the total memory requirement. This algorithm achieve high memory efficiency, and low memory bandwidth requirement.

The future work is to extend the algorithm to have level-based skip-distances and number of prefixes per node for different trie levels.

5. REFERENCES

[1] Hoang Le, Weirong Jiang Juniper, Viktor K. Prasanna, Memory-Efficient IPv4/v6 Lookup on FPGAs Using Distance-Bounded Path Compression. IEEE International Symposium on Field-Programmable Custom Computing Machines, 2011, 242-249.

[2] H. Le, W. Jiang, and V. K. Prasanna. A sram-based architecture for trie-based ip lookup using fpga. In *Proc. FCCM '08*, 2008.

[3] M. Behdadfar, H. Saidi, H. Alaei, and B. Samari. Scalar prefix search - a new route lookup algorithm for next generation internet. In *Proc. INFOCOM '09*, 2009.

[4] D. R. Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15(4):514–534, 1968.

[5] K. Sklower. A tree-based packet routing table for berkeley unix. In *Winter Usenix Conf.*, pages 93–99, 1991.

[6] F. Baboescu, D. M. Tullsen, G. Rosu, and S. Singh. A tree

based router search engine architecture with single port memories. In *Proc. ISCA '05*, pages 123–133, 2005.

[7] A. Basu and G. Narlikar. Fast incremental updates for pipelined forwarding engines. *IEEE/ACM Trans. Netw.*, 13:690–703, June 2005.

[8] H. Song, M. S. Kodialam, F. Hao, and T. V. Lakshman. Scalable ip lookups using shape graphs. In *Proc. ICNP '09*, 2009.

[9] I. Sourdis, G. Stefanakis, R. de Smet, and G. N. Gaydadjiev. Range tries for scalable address lookup. In *Proc. ANCS '09*, 2009.

[10] M. Wang, S. Deering, T. Hain, and L. Dunn. Non-random generator for ipv6 tables. In *HOTI '04*, pages 35–40, 2004.

[11] R. Sangireddy, N. Futamura, S. Aluru, and A. K. Somani. Scalable, memory efficient, high-speed ip lookup algorithms. *IEEE/ACM Trans. Netw.*, 13(4):802–812, 2005.

[12] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network*, vol. 15, no. 2, pp. 8-23, 2001.

[13] H. Le and V. K. Prasanna. Scalable high throughput and power efficient ip-lookup on fpga. In *Proc. FCCM '09*, 2009.

[14] A. Andersson and S. Nilsson. Efficient implementation of suffix trees. *Softw. Pract. Exper.*, 25:129–141, February 1995.