Journal of Advances in Computer Research
Quarterly ISSN: 2008-6148
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 3, No. 3, August 2012), Pages: 19-34
www.jacr.iausari.ac.ir

A High Performance Parallel IP Lookup Technique Using Distributed Memory Organization and ISCB-Tree Data Structure

Mahmoud Hasanloo[1]✉, Ali Movaghar[2]

(1) Department of Electrical and Computer Engineering, Zanjan Branch, Islamic Azad University, Zanjan, Iran

(2) Computer Engineering Department, Sharif University of Technology, Tehran, Iran

hasanlou@gmail.com; movaghar@sharif.edu

Received: 2012/05/10; Accepted: 2012/06/30

Abstract

The IP Lookup Process is a key bottleneck in routing due to the increase in routing table size, increasing tra□c and migration to IPv6 addresses. The IP address lookup involves computation of the Longest Prefix Matching (LPM), which existing solutions such as BSD Radix Tries, scale poorly when tra□c in the router increases or when employed for IPv6 address lookups. In this paper, we describe a high performance parallel IP lookup mechanism based on distributed memory organization that uses P processor for solving LPM problem. Since multiple processors are used, the number of prefixes to be compared for each processor has been reduced. In other words each processor needs to find LPM for a specific IP address among N/P of prefixes. In order to reduce the number of memory access in each processor which is a major bottleneck in IP lookup process, we use ISCB-Tree data structure for the sake of storing the forwarding table in each processor. ISCB-Tree is a B-Tree like data structure that reduces the height of prefix tree and logarithmic growing manner with the increasing number of prefixes. By the using of this data structure the number of memory access reduces sharply.

Keywords: *IP lookup, Packet forwarding, ISCB-Tree, Router organization, Parallel processing*

1. Introduction

Due to exponential growth in the number of users (hosts) and applications of internet, the network traffic is increasing daily. In order to satisfy users need we need two main components in network. First is addressing and second one is high performance core network. In the internet we should associate a unique IP address to each host. With regard to numerous numbers of hosts, further IPv4 is not enough to addressing. The addressing problem has been solved with the advent of IPv6 which is 128 bits in length. With IPv6 we can address 2^{96} times more hosts than IPv4. But it means many more and longer prefixes in backbone routers which we will discuss about this problem later in this section.

The second problem is high performance core net-work for the sake of satisfying increasing needs of users' applications to faster downloads and uploads.

Core network consists of two main components: links and routers. Thereupon we should use high speed links and faster routers. The former was satisfied by using of fiber optics for now. But faster routers are still unsolved problem.

In the network layer a router has two main functions: routing and forwarding. Routing is a process to construct a table which tells us in order to reach to a network from which port a packet should be forwarded. This table which is called forwarding table, stores myriads of tuples in the form of prefix/port format. The packet forwarding process in a router involves finding the prefix in the forwarding table that provides the longest match to the destination address of the packet to be routed.

When an IP router receives a packet, it must search for the best prefix in its forwarding table that has the longest match when compared to the destination address in the packet. The packet is then forwarded to the output link associated with that prefix. Because number of existing prefixes in a routing table of a backbone router is about 400k and it is increasing daily, this process of finding the longest prefix match (LPM) is one of the bottlenecks in the packet forwarding process. With regard to problems resulted from using IPv6, more and longer prefixes in the forwarding table, this process become more complicated even more.

For example, suppose a backbone router with capacity of 50Gbps which forwards IP packets with typical length of 5kb. With a simple calculation ($50 * 109/5 * 103$), we find this router should do IP lookup process 50 million times per second. In other words a backbone router should search for LPM among 400k prefixes 50 million times per second, which is a huge processing.

As we can imagine number of memory accesses has vital effects in this process. Thus we should use a data structure which reduces number of memory accesses for a LPM search. In the most of well-known database management systems (DBMS), B-Tree data structure and its variations are used to store tuples. As we will see soon it is not proper for storing prefixes however we can customize it for our usage. We propose an IP lookup specific customized B-tree (ISCB-tree) to store prefix set.

With regard to mentioned simple example, we can inference a single processor cannot do this amount of processing per second. So in this paper we propose a parallel processing schema using P processors based on distributed memory organization which uses ISCB-Tree data structure to store prefixes in order to do IP lookup faster and faster.

The rest of the paper is organized as follows. In section 2 we review some of the major methods to IP lookup. Section 3 describes our lookup schemes. Sub-section 3.1 describes the ISCB-Tree data structure and its search, insert, and other procedures for maintaining this data structure; Subsection 3.2 describes the partitioning mechanism of routing table. Subsection 4.1 describes performance measurements in terms of queuing theory and subsection 4.2 illustrates simulation results of our scheme. Finally, Section 5 concludes the paper.

2. Related Work

More sophisticated and efficient approaches have been introduced in several works in which a suitable data structure named forwarding table, is constructed from the routing table. Proposed methods for IP Lookup can be categorized based on their five properties, platforms, data structures, processing methods, forwarding table partitioning methods and caching.

Based on platform, schemas can be classified in two broad categories: Hardware based methods [22, 4, 7, 11] which are almost very fast but are not as well scalable and software based methods [14, 27, 24, 29] which typically is scalable but not as fast as hardware based methods. We will focus on software based methods in this paper.

Data structure is a vital component in a schema because it has main effect in number of memory accesses for finding LPM. When we use a data structure we should notice to its scalability as well as its depth. Almost each level of data structure indicates single memory access. IP lookup methods, based on their used data structure for the sake of forwarding table construction can be classified into three main categories: 1) Table based [26]; 2) Trie-based [24, 19]; and 3) Tree-based algorithms [17, 15, 16].

Our mean from processing method is that does a method use parallel processing or not? Most of old methods used single processor but with regard to growing forwarding table size and prefixes length in IPv6 it seems there is no way other than using several processors in a parallel manner to satisfy today routers' needs.

Most of parallel processing methods partition forwarding table and distribute them among several memory modules in order to reduce number of prefixes that LPM should found among them. The partitioning schemas can be categorized in two broad classes: 1-those that associate a special memory module to each processor in which a segment of forwarding table is stored. These schemas typically split forwarding table based on few bits of prefixes and when a packet received in the router, the desirable processor is indicated with regard to those bits. 2-Those that each processor can access all of memory modules. These schemas typically use PRAM memories to handle several requests from different processors.

Finally some works, for example [28], cache an LPM when it found in order to use it for next packets of the same flow to speedup lookup process and in spite some others [14] eliminated cache from their schema because of stolen tuples which may exist in cache. Also some others exist which the cache is base of their schema[21].

Following we describe shortly some of done works in IP lookup fields in the past decade. And finally in this section we will describe our schemas important properties to show its effectiveness.

Presently many IP-Lookup mechanisms are proposed for IPv4 routers but most of them cannot be scaled very well for IPv6. Data structure used by most of them [24, 26, 19] become very large when applied for IPv6, so the number of memory access for these mechanisms grow exponentially. Data structures of some mechanisms [15, 16] become very large for IPv6; however they cannot be as fast as we need today. For these reasons researchers have become interested in using parallel processing for IP lookup [11, 3, 9, 6, 5, 8]. A parallel IP lookup technique must perform two jobs in order to increase lookup speed: 1) split forwarding table to some smaller sub-tables for reducing the number of prefixes which lookup must be done among them; and 2) use several processors for speeding up lookup process.

In [6] authors partition the routing table based on four bits, called ID bits, of prefixes into sixteen section and associate a processor to each of them, then when a packet is received, based on its destination address ID bits a processor is responsible to find LPM for this packet. In this mechanism at most sixteen IP address can be looked up simultaneously. In this technique each processor works on different set of prefixes but not all of processors work at the same time, in the best case all of them work simultaneously. We know that packet incoming has burst manner thus their incoming distribution is not normal based on their ID bits. Also contriving of sixteen processors

on a board is not so simple, at least because of their heating problem and if want to use several boards for processors then the speedup of this mechanism will be impressed with the communication cost of processors.

In [27] authors proposed an IP lookup technique based on CREW PRAM. They sort prefixes and split routing table into subsequences of equal length among P processor. When a packet is received, its destination address broadcasts to all processors then each of them compute LPM for this packet simultaneously and finally the main LPM is found by comparing LPMs of all processors. The mechanism uses CREW PRAM memory in order to establish a communication among processors in the broadcasting phase. As we said previously memory access is a major bottleneck and time consuming action in the IP lookup process and all trying to reduce the number of memory accesses. So increasing the number of memory accesses by one is not an efficient way in order to obtain destination address of packet since the number of memory accesses is less than ten in traditional mechanisms.

In some other works like [11] authors propose partitioning of forwarding table and some processors work simultaneously to find LPM for a packet, then the results are compared together and the best one is chosen.

In [21] authors present a cache-based IP lookup technique. They cache prefixes rather than full IP address when a lookup is done. In this schema prefixes are divided into two categories: 1-non-cacheable prefixes and 2-cacheable prefixes. Non-cacheable prefixes are those that encompass other prefixes and remaining prefixes are cacheable. In order to increase number of cacheable prefixes, Kasnavi and et. al. expanded prefixes shorter than 16 bits in length. When an IP address matches with a cacheable prefix, it will cache in TCAM memory, otherwise full IP address will be cached. They cache prefixes with hope that it will match incoming packets destination address with regard to locality in internet traffic. Trie is used in this schema as a data structure in which prefixes are stored.

Authors of [28] introduced a parallel architecture for IP lookup. They used tire data structure to store prefixes, then trie is partitioned into disjoint sub-tries using initial bits of prefixes. They use an efficient algorithm to distribute sub-tries among memory modules such as they have nearly equal content. The other important technique which they used is early caching, which allows the destination IP address of a flow to be cached before its next-hop information is retrieved. When a matching is done, the result is used to forward all of the existing packets in the system for the same flow.

Authors of [18] assume each prefix as a range such as $[b_r, e_r]$ and store B_Rs at leaf nodes of B-Tree like data structure as keys. They have a rule about prefix storing which tells Store a prefix r at a node or leaf key V if and only if the span of V is contained in $[b_r, e_r]$ but the span of the parent of V is not contained in $[b_r, e_r]$. The search algorithm is much like B-Tree search. They tried to reduce update time in beside of fast lookup time by using M-way Tree data structure.

A heap like data structure is built in [10], in which the longest existing prefix become the root and others fill out remaining nodes such that longer prefixes stay in upper layers. When an IP packet is received, search process began from the root and the first matching prefix will be the longest one because of the mentioned used concept in data structure construction.

Chang in [29] expanded all prefixes to n bits and store them in a hypercube to simulate binomial spanning tree. To find LPM it has a trie like manner in other words it find LPM by getting bits of destination address and go ahead among hypercube nodes.

First of all authors of [23] sort prefixes based on their integer values. Then, prefixes are categorized based on n first bits, which n is the longest common substring among prefixes, and these bits are stored in an index table. This process continually is applied on remaining bits of prefixes until a multi-level index tables are achieved.

An efficient algorithm is presented in [13] to construct shape-shifting trie (SST) data structure. SST is a trie based data structure in which each node has at most k nodes of trie. They try to construct this data structure with minimum height which is improve worst-case lookup time.

In [12] a parallel architecture with the client-server model is presented in which one processor receives all incoming packets and by two first bytes of their destination IP address locates the LPM region boundary in the prefix table which is stored in a shared memory module. Then it sends this boundaries and destination IP address to one of the processors. Finally the receiver processor does the binary search among sorted prefixes with regard to received boundaries and destination address.

The current work is different with all others in the following manner:

- A parallel architecture is used in which all processors can work simultaneously to lookup destination addresses of several packets concurrently.
- An efficient data structure (ISCB-Tree) based on famous B-Tree is proposed for the sake of prefix table construction which stores prefixes such that the height of the tree is small for large number of prefixes and its scalable very well. This reducesnumber of memory accessessharply which is the most time consuming element of the lookup process.
- The ISCB-Tree is partitioned simply to several parts and each of them stored in a multiple simple SRAM memory modules instead of PRAMs. This reduces the cost of memory management subsystem of the router and increases effective memory access time.
- In addition the above strengths, early caching technique which originally was proposed in [28] are used. By using this technique the number of IP address which should be looked up is reduced and finally overall speed of lookup process is increased.

3. The Proposed Scheme

Our proposed router has two main components: 1) data structure which used to construct forwarding table, in order to improve lookup process by reducing the number of memory accesses, and 2) parallel architecture which splits forwarding table into sub-tables and assign a processor to each of them. Splitting of forwarding table reduces the number of prefixes that the search algorithm should look among them to find LPM of an incoming IP packet.

3.1 Data Structure

As we know in almost well-known database management systems (DBMS), which they are symbol of data storing and retrieving, B-tree [25] and its variations are used in order to speedup data manipulation operations. So in the current work this valuable experiment is regarded and preferred to use B-Tree data structure in order to construct forwarding table which contains huge number of prefixes and needs very fast search for IP lookup operation.

Table1. A sample forwarding table

Prefix	Port
00110*	3
0011*	2
10110*	1
10111*	1
1001*	1
10011*	2
101100*	3
101101*	4
1110*	2
0110*	2

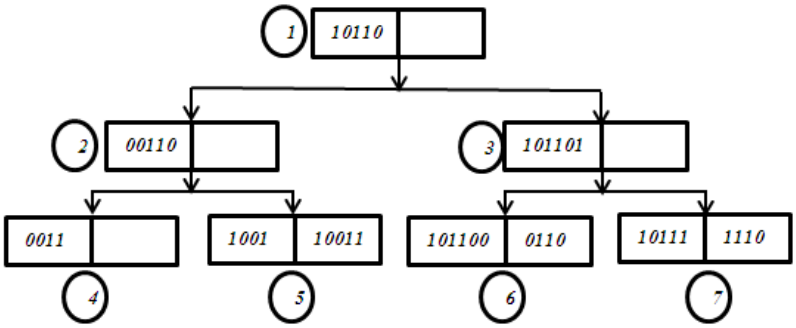

Figure1. B-Tree of prefixesin table1

3.1.1 Using B-Tree data structure to construct forwarding table

A B-tree is a data structure that maintains an ordered set of data elements and allows efficient operations to find, delete, insert, and browse them. In this discussion, each piece of data stored in a B-tree will be called a ”key”, because each key is unique and can occur in the B-tree in only one location.

A B-tree consists of “node” records containing the keys, and pointers that link the nodes of the B-tree together. Every B-tree is of some “order n”, meaning nodes contain from n to 2n keys, and nodes are thereby always at least half full of keys. Keys are kept in sorted order within each node. A corresponding list of pointers is effectively interspersed between keys to indicate where to search for a key if it isn’t in the current node. A node containing k keys always also contains k+1 pointers.

For the sake of B-Tree construction, a comparing method is neededto identify the relative position of each prefix in the set of prefixes. So the following definition is used which is most like dictionary compare method.

Definition 1: Prefix Comparison: Suppose A = $a_1..a_n$andB = $b_1..b_m$ are two prefixes. If m=n then numerical values of prefixes are compared to determine which prefix is bigger. Otherwise, suppose m < n; numerical values of A = $a_1..a_m$andB = $b_1..b_m$ are compared. Prefix with larger value is considered to be larger. If A = $a_1..a_m$andB = $b_1..b_m$ are identical, then, prefix A considered as a larger one because it’s longer than B. Table 1 shows an example forwarding table and in figure 1 we can see corresponding B-tree. Following with an example it will be shown why B-tree cannot be used in IP lookup methods. Suppose a packet received to the router with destination IP address 00111100 (For simplicity, in this example the IP address length is assumed to be eight bits). Now

the search method compare this value with prefixes in the root node and navigate us to the left side namely node 2. The IP address does not match with prefix 00110* which is the only key in the node 2 and also is greater than it, so search will be continued from node 5. There is no matching prefix and router will decide to send the received packet to the default gateway(e.g via port 1). Whereas we have prefix 0011* in node 4 which matches with this address and it indicates that this packet should be forwarded via port 2.

This problem occurs because there exist two prefixes 0011* and 00110* which the former is prefix of the later. We should attend that if there is 00111* exists instead of 00110* yet this problem exists from other direction. We have following definitions for this situation.

Definition 2: Enclosure: A string S is called an enclosure if there exists at least one data string A, such that S is a prefix of A. A is called an enclosed data element. For example 0011 is an enclosure of 00110.

Definition 3: Disjoint prefixes: for any two different prefixes A and B, we say Aand B are disjoint if and only if A is not enclosure of B and vice versa.

3.1.2 IP lookup Specific Customized B-Tree (ISCB-Tree)

Just like traditional B-Tree in ISCB-Tree each node has K keys and K+1 pointers in each node. While in B-Tree all keys are similar in ISCB-Tree there exists two types of keys. Each key field type in this tree is either typical or special type.

If enclosure prefixes stay in upper layers than the enclosed prefixes, this problem will be solved. Authors of [14] used this solution in construction of their B-Tree and called resulted Tree as DMP-Tree. But DMP-Tree has following restrictions:

- DMP-tree does not guarantee minimum node utilization.
- It is not possible to guarantee that the final tree is balanced.

Typical key type (type T) is used for disjoint prefixes storing and has not any other thing than B-Tree keys. But special key types (type S) are used to store enclosure prefixes. It has an additional pointer which point to another ISCB-Tree.This tree contains enclosed prefixes of that enclosure prefix. Following main operations of ISCB-Tree are described.

A. Search:search method begins from the root node and compares incoming IP address with pre-fixes in the node to find P_i, such that $P_i < IP <= P_{i+1}$. Three states may occur in the search result.

1. $P_i$andP_{i+1} don't match IP address then the pointer between P_i and P_{i+1} is followed to the next level node.
2. If$P_i$orP_{i+1} match IP address and it is an S type key then this matching is saved and its pointer is followed to sub-ISCB-tree with hope that a longer matching may be found in this sub-tree.
3. $P_i$orP_{i+1} match IP address and it is a T type key, this is the LPM.

This process continues until an LPM found or following pointer becomes null (A leaf or a terminated tree path). In the latter case the packet will be forwarded through default route.

B. Insertion: insertion is the most important part of the building process of ISCB-tree. First of all the search method initiated to find the proper location of new prefix. In the search algorithm two states may occur as follow:

1. The new prefix is a disjoint or enclosed prefix. In this case search algorithm navigates us to a leaf node which the new prefix should be inserted. The new prefix is inserted in the proper position at the current node. If the node is full then it should be divided into two new nodes with half full capacity.
2. The new prefix may be enclosure of some of other prefixes. In this case, any enclosed prefix along the path is removed (by delete algorithm) and added to sub-tree of the new S-type key. Also all sub trees of the first detected enclosed prefixshould be searched for the sake of finding other probable enclosed prefixes which may exist.

In the latter case insertion time is proportional to the number of enclosed prefixes which exist in the mentioned sub-tree. In order to reduce this time one of the following solutions can be used:

1. Sort prefixes before construction algorithm is begun.
2. Classify existing prefixes and detect all disjoint, enclosure and enclosed prefixes.

C. Deletion: The delete algorithm is very similar to the insert, just a little more complicated. It also requires that we use the basic ideas from the search algorithm to locate the proper leaf, then remove (in this case) the item, and readjust the keys in the leaf.

However the deletion candidate key may be T or S type key. If the key is T type simply remove it from the node else in addition to prefix deletion we should insert (using insert algorithm) sub-tree enclosure or disjoint keys again in the tree. Now it is possible that the leaf is less than half full. It must work with its siblings to restore the structure of the tree. If it has a sibling with extra keys, a key can be moved from one leaf to the next (with the appropriate changes in the parent node).

On the other hand, if the sibling doesn't have any keys to spare, then the two nodes must be merged. This will obviously require that the parent node give up a key (since it no longer has to distinguish between the two leaves). This might make the parent too small as well, and necessitate further consolidation up the chain.

The following example shows the building process of ISCB-Tree using data items in table 1. In this example branching factor is set to three, it means that each node at most contains two data elements and has three children. Also there is an assumption that data items will be added to the tree with the same order shown in the table 1.

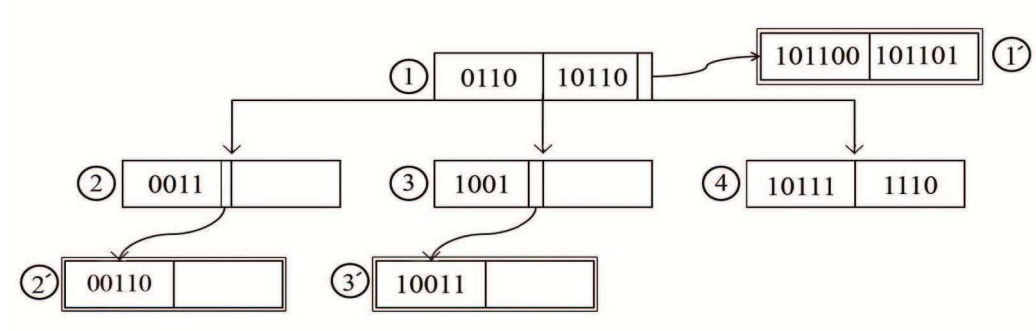

Figure2. ISCB-Tree of prefix of table1

Initially tree is empty then the first item, 00110 is added to the first and the root node of the tree. When the second item namely 0011wants to added to the tree, it's

recognized that this is an enclosure of the first item. So 0011 is added to the tree as an S-type key and 00110 is added to its sub-tree. Now is the turn of third item named 10110 to be added, since it is a disjoint prefix, simply is added to the node. Inserting the fourth prefix, 10111, causes overflow and the node must be split. The median, 10110, is chosen as the splitting point. Figure 2 illustrates the final ISCB -Tree created for items of table 1.

Now by an example the search algorithm will be clarified. Suppose as B-tree example a packet arrived and its destination address is 00111100. Algorithm takes the root node, node 1, and compares its elements with this address. This address is smaller than the first element, 0110, so it follows the left most link and reaches the second level in the tree to node 2. In the second level, this address is match with the only existing element namely 0011. But this key is an S-type key. So the algorithm saves this matching and follows the sub-tree link of this element and reaches to node 2′. In the sub-tree it does not match with the 00110 so algorithm takes the saved element, 0011, as LPM for this address. As it can be seen in table 1, it is a true choice.

The most important strengths of the ISCB-Tree can be summarized as follow:

1. The height of tree is as small as possible with regard to node utilization and tree balancing which are properties of B-Tree. So the number of memory accesses needed to find an LPM is much fewer than traditional used data structures like trie.
2. The first found prefix is the LPM if the key is not S-type key.
3. The main tree balancing is guaranteed as in B-Tree
4 Search method is very simple

3.2 Parallel Architecture

Our proposed Multiprocessor based router contains up to N processors and N memory modules. Each memory module is assigned to a processor. A sample structure with 8 processors is illustrated in figure 3. As it can be seen in this figure the architecture consists of flow table, queue table, index table, processors and memory modules. In the following each of these components will be described.

As shown in the previous sub-section the set of prefixes are stored in an ISCB-Tree. In this mechanism, the ISCB-Tree is partitioned into sub-trees and each of them stored in a separate memory module. Suppose the used branching factor for the ISCB-Tree is eight (seven key per node), so it has eight sub-trees in root node. In this architecture, one can use eight memory modules and store each of sub-trees in one of them. It's remarkable that root node prefixes are stored again in the sub-trees.

An index table is built by using root node prefixes and used in order to detect which search engine (a memory module with its assigned processor called search engine) should do the IP lookup process for an incoming packet. These prefixes are expanded by adding zeros to their tail to make them as long as possible (32 bits for IPv4 and 128 bit for IPv6). Then they stored in the index table with ascending order (the order which they was stored in the root node of ISCB-Tree).

Now destination address of an incoming packet can be compared with values of index table to determine the proper search engine. For example if destination address is greater than the index[0] and less than the index[1] values, the proper search engine for this IP is number 1 (with the start number 0). When the search engine is determined, the packets destination address will save in a slot of flow table in ascending order. Also the packet will store in the same slot of the queue table.

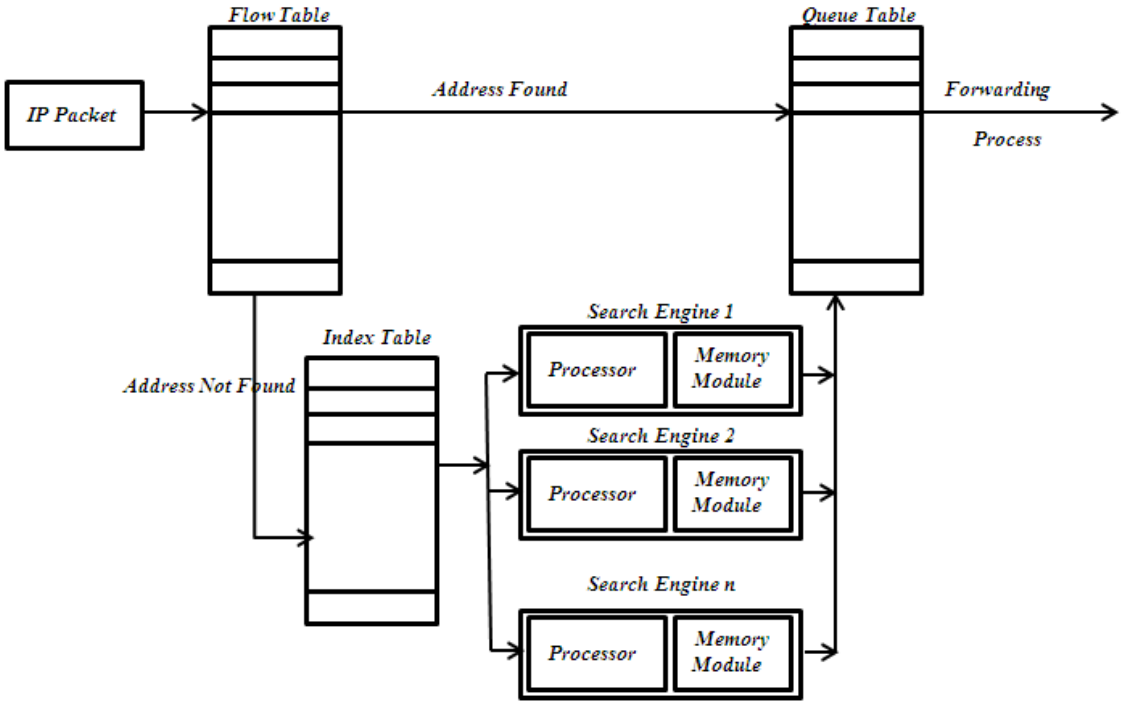

Figure3. Parallel Architecture of our Schema

Flow and queue tables are used to incorporate a caching method. Similar to our caching method was introduced in [28] and called early caching. But our schema much like to caching method of solution that was explained for reader-writer problem in [1]. However this caching method is also called early caching.

When an IP packet received at the router, its destination address will be searched in the flow table. If it does not exist then the destination address registered in this table and go to a proper search engine to find LPM. As it mentioned previously the desired search engine is determined by using the index table. But if the IP address exists in flow table, it means that a packet of the same flow reached to the router and went to a search engine before this one. So it does not need do the same LPM search for the new packet. Therefore packet will go to the queue table and wait for its predecessor to find LPM. Queue table is a 2-D array which stores packets of each active flow in a row.

By the receiving an IP address in a search engine, the processor will begin search algorithm to find LPM for that address in its own memory module. As soon as LPM is found, processor 1) removes index from flow table; and 2) forwards all of the waiting packets of that flow in the queue table. Indexes are removed from flow table as soon as possible in order to avoid stale entries.

The main strengths of this structure are as follow:

1. Reducing the number of prefixes which should be searched to find LPM of an IP address by factor of 1/N (N is the number of search engines)
2. N IP address can be simultaneously searched.
3. This structure uses cheap equipment. In other words SRAM memory modules can be used instead of TCAM memories which most of parallel architectures used.

4. Early caching speed up packet forwarding while we avoid stale cache entries which may occur in almost previously proposed caching architectures.

4. Performance Measurement

In this section first we model our proposed mechanism by M/M/1 queue and solve the model in order to illustrate performance of proposed mechanism in terms of input rate, service rate, and etc. parameters. Then in subsection 4.2 we report simulation results that show the performance of our mechanism in terms of number of memory access.

4.1 Queuing System Analysis

We useM/M/1 queuing model in order to model IP lookup parallel architecture. We see this multiprocessorsystem as N single processor subsystem. By assuming the normal distribution of packet incoming for all sub-Branching Factor systems, the mean arrival rate of each subsystem is 1/N of the mean arrival rate of multiprocessor system with N processors.

We assume that the arrival process of the incoming IP addresses is in Poisson distribution with a mean arrival rate as λ_0. The mean service rate for the incoming IP addresses in each processor is considered to be μ_0. So the overall arrival rate is $\lambda=N*\lambda_0$ and overall service rate is $\mu=N*\mu_0$.

Using the queuing theory to solve the model performance measures, such as the utilization factor (ρ), probability of the search engines queue to be empty (P_0), probability of being K IP address in router (P_K), probability that an incoming IP address has to wait in the queue (P_Q), average number of IP addresses waiting in the queue to be processed (N_Q), and mean response time (T) were found.

- Utilization factor
  - Utilization factor for each search engine: $\rho_0 = \frac{\lambda_0}{\mu_0}$
  - Overall utilization factor: $\rho = \frac{N*\lambda_0}{N*\mu_0} = \rho_0$
- The probability that the system is idle: $P_0 = 1 - \rho$
- The probability of presence of K IP packet in router: $P_k = (1 - \rho)\rho^k$
- The probability that an incoming packet has to wait in queue: $P_Q = 1 - P_0 - P_1 = \rho^2 + 2\rho$
- The average number of packets in router: $N_Q = \frac{\rho}{1-\rho}$
- Mean response time: $T = \frac{1}{\mu-\lambda}$

4.2 simulation Results

At first a set of simulations were conducted in order to obtain optimal branching factor for ISCB-Tree when data items are IPv6 prefixes, then another set of simulations were conducted to determine efficiency of our parallel algorithm with increase in number of prefixes existing in forwarding table.

Simulation results, figure 4, show that the optimal branching factor is between six to nine in which the height of tree reduces to between six and eight. After this point height of the tree does not reduces so much with increasing of branching factor. As we can see in in figure 4 the height of ISCB-Tree growths in logarithmic manner with respect to the

number of prefixes, so our algorithm can be scaled very well with growing the size of the forwarding tables.

When a node of tree becomes large, processor cannot read whole of the node in one memory access so increasing of branching factor may causes more than one memory access for reading of a node. Our simulation results show that the branching factor of eight is the optimal for most of data sets. So we use this branching factor in all other simulations.

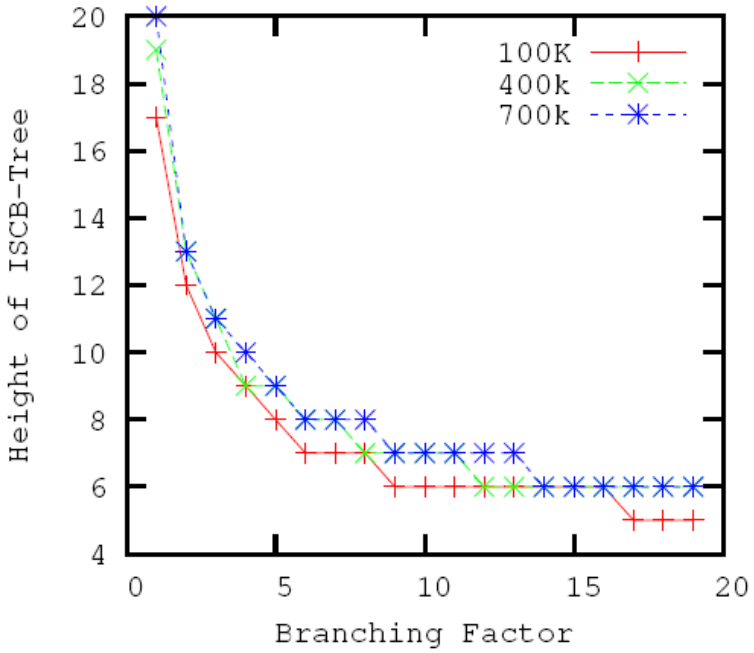

Figure4. ISCB Maximum Height

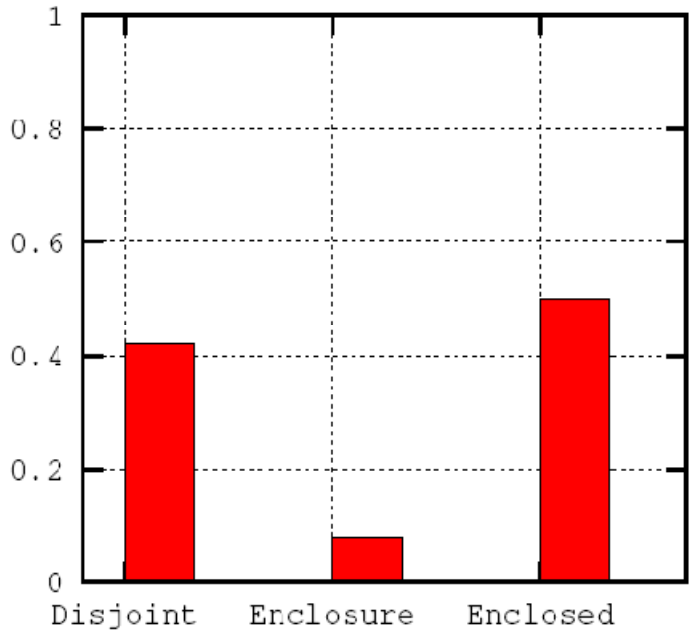

Figure5. Statistics of Enclosure, Enclosed and Disjoint Prefixes

Analysis of routing tables where downloaded from [2] shows that about half of prefixes are enclosed prefixes, figure 5, when they are moved to sub-trees, the height of main tree becomes as small as four levels with branching factor eight for 130k of prefixes. Also number of enclosed prefixes per enclosure is between 1 and 1262 which distribution of them depicted in figure 6. As it can be seen in this figure more than 81% of enclosure prefixes have less than eight prefixes which mean most of sub-trees have only single level.

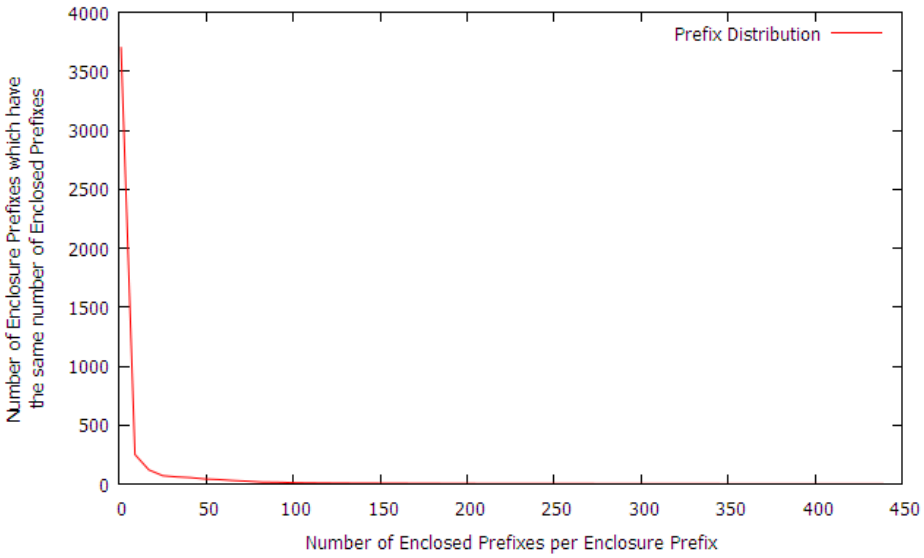

Figure6. Distribution of Enclosed Prefixes

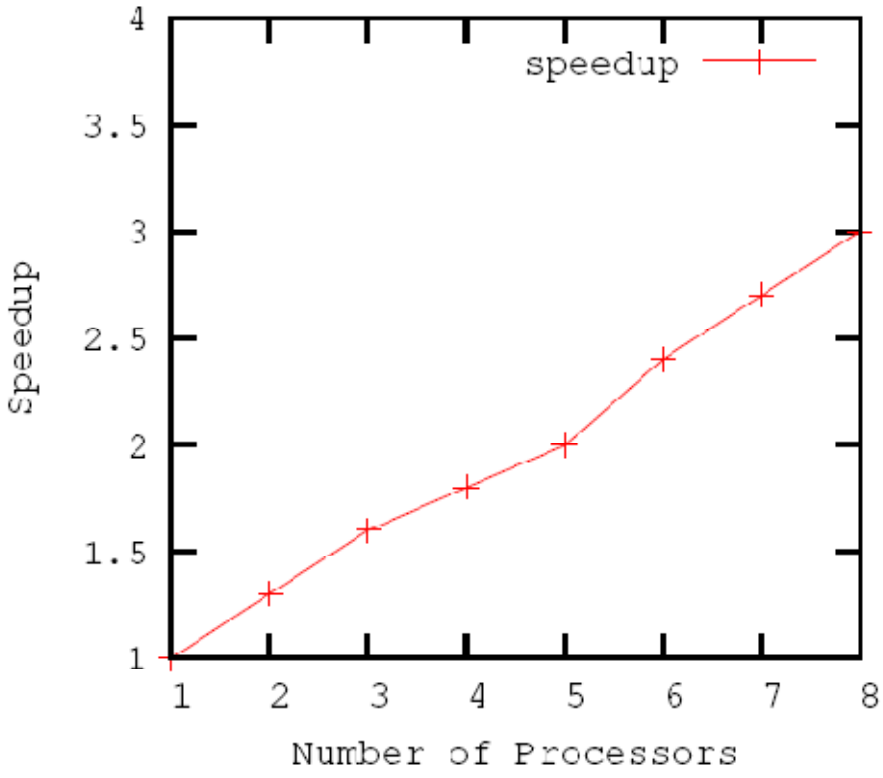

Figure7. Speedup of Parallel Architecture

We simulate our proposed method with 100k, 400K and 700k of prefixes. We download some routing table from [2] and based on distributions of current prefixes, generate some others in order to achieve the 700k of prefixes. When we construct ISCB-Tree and divide it based on the root node into eight categories each of them has 87k ± 6k of prefixes.

Our simulation results show that with branching factor eight and a data set of size 90k for each processor (and 700k in overall) the average number of memory access is reduced to 4.1 since with single processor this number is about 7.4. This number of memory access in our mechanism is much better than the current algorithms.

A good measure for analyzing the performance of a parallel algorithm is speedup. Speedup is defined as the running time of sequential algorithm and the running time of parallel algorithm.

In order to determining the efficiency of our method we ran several simulations with different size of data sets. Figure 7 shows the speedup of our proposed mechanism. As we can see in this figure by increasing the number of prefixes, the speedup becomes more sensible.

5. Conclusions

We simulate our proposed method with 100k, 400K and 600k of prefixes. We download some routing table from [2] and based on distributions of current prefixes, generate some others in order to achieve the 700k of prefixes. When we construct ISCB-Tree and divide it based on the root node into eight categories each of them has 97k ± 6k of prefixes.

The major time consuming step in lookup process is finding LPM. In order to obtain better performance in this step we must try to reduce the number of memory accesses which is the major bottleneck in lookup process.

In this paper we presented a new data structure called IP lookup Specific B-tree (ISCB-tree) based on the most well-known B-Tree data structure. ISCB-Tree is used to construct forwarding table besides using of B-tree strengths. ISCB-Tree reduces number of memory accesses sharply since the height of the tree has the main effect on number of memory accesses for an LPM search. The height of ISCB-Tree growths in logarithmic manner with respect to the number of prefixes, so our algorithm can be scaled very well with growing the size of the forwarding tables.

Also in this paper we proposed a parallel architecture that partition forwarding table among N processors, which N is optional but it’s better to be chosen same as branching factor of ISCB-Tree. By this partitioning, we store each sub-tree in a memory module and assign a processor to each of them. We call this package search engine which contains a sub-tree, a memory module and a processor.

When an IP packet received in the router, its destination address will be searched in a flow table. If exists an entry for that destination address in the flow table, this packet goes to the queue table and waits for search result for its descendant. Else destination address will be inserted to the flow table and by using an index table the appropriate search engine is determined for this packet. Then the selected search engine finds LPM for the given address and forwards all packets for this destination which are waiting in the queue table.

We have also simulated our proposed mechanism using the real forwarding tables that its results show that our mechanism has better performance in comparing with present parallel mechanisms.

It must be noted that our mechanism obtains this performance using few processors. Also we must notify that contriving big number of processor, like sixteen of processors, on a small board is not a simple task, so the mechanisms that use big number of processors are not highly practical.

6. References

[1] A. Silberschatz, P. B. Galvin, G. Gagne, Operating System Con-cepts,John Wiley & Sons, 7th edition, 2004.

[2] BGP reports, http://bgp.potaroo.net/, july 2011.

[3] G. Bongiovanni, P. Penna, XOR-based schemes for fast parallel IP lookups, Proceedings of the 5th Conference on Algorithms and Complexity 2003 (CIAC ??03).

[4] H. Mohammadi, N. Yazdani, Robatmili, B., and Nourani, M., HASIL: Hardware Assisted Software-based IP Lookup for Large Routing Tables, Proceeding of the 11th IEEE Interna-tional conference on networks 2003 (ICON 03).

[5] J. Wang, K. Nahrstedt, Parallel IP Packet Forwarding for To-morrows IP Routers, Proceedings Of IEEE Workshop on High Performance Switching and Routing 2001 (HPSR ??01).

[6] K. Venkatesh, S. Aravind, R. Ganapath, T. Srinivasan, A High Performance Parallel IP Lookup Technique Using Distributed Memory Organization, Proceedings of the International Confer-ence on Information Technology: Coding and Computing 2004 (ITCC 04).

[7] K. Zheng, C. Hu, H. Lu, B. Liu, An Ultra High Throughput and Power E□cient TCAM Based IP Lookup Engine, Proceedings of IEEE INFOCOM 2004.

[8] K. Zheng, H. Lu, B. Liu, A Parallel IP Lookup Algorithm for Terabit Router, Proceedings of IEEE ICCT 2003.

[9] L. Hyesook, L. Bomi, A New Pipelined Binary Search Architec-ture for IP Address Lookup, Proceedings of IEEE HPSR 2004.

[10] L. Wuu, T. Liu and K.Chen, A longest prefix first search tree for IP lookup, Elsevier, Journal of Computer Networks 51, pages 3354-3367, 2007.

[11] M. Akhbarizadeh, M. Nourani, An IP Packet Forwarding Tech-nique Based on Partitioned Lookup Table, Proceedings of IEEE ICC 2002.

[12] M. Hasanloo, M. Fathi, A. Amiri, A High Performance Parallel IPLookup Technique Based onMultiprocessor Organization and CREW PRAM, Second Asia International Conference on Modelling& Simulation 2008.

[13] M. Pan, H. Lu, Build shape-shifting tries for fast IP lookup in O(n) time, Computer Communications 30, pages 3787-3795, 2007.

[14] N. Yazdani, H. Mohammadi, "DMP-Tree: Dynamic M-way Prefix Tree Data Structure for String Matching", Article in press, Elsevier Computers & Electrical Engineering,Volume 36, Issue 5, September 2010, Pages 818-834.

[15] N. Yazdani, H. Mohammadi, DMP-Tree: Dynamic M-way Pre-fix Tree Data Structure for String Matching, Elsevier journal of Electrical Engineering and Computer Science 2007.

[16] N. Yazdani, H. Mohammadi, IP Lookup in Software for Large Routing Tables Using DMP-Tree Data Structure, Proceeding of the 9th Asia Pacific Conference on Communications 2003 (APCC 03).

[17] N. Yazdani, P. Min, Fast and Salable Schemes for IP Lookup Problem, Proceeding of IEEE Conference on High Performance Switching and Routing 2000 (HPSR 2000).

[18] P. Warkhede, S. Suri, G. Varghese, Multiway range trees: scal-able IP lookup with fast updates, Elsevier, Computer Networks 44, pages 289-303, 2004.

[19] P. Yilmaz, A. Belenkiy, N. Uzun, A Trie-based Algorithm for IP Lookup Problem, Proceedings Of ACM SIGCOMM 1997.

[20] R. Bayer, C. McCreight, Organization and Maintenance of Large Ordered Indexes, Acta, Informatica 1,3, 1972.

[21] S. Kasnavi, P. Berube, V. Gaudet, J. Amaral, A cache-based internet protocol address lookup architecture, Journal of Com-puter Networks 52, pages 303-326, 2008.

[22] S. Kaxiras, G. Keramidas, IPStash: a Power-E□cient Mem-ory Architecture for IP-lookup, Proceedings of the 36th In-ternational Symposium on Microarchitecture 2003 (MICRO-36 2003)
[23] S. Leu, R. Chang, A fast and scalable IPv4 and IPv6 ad-dress lookup algorithm, Elsevier, Computer Communications 29, pages 36020-3036, 2006.
[24] S. Sahni,K. Kim,E□cient Construction Of Variable-Stride Multibit Tries For IP Lookup, Proceedings IEEE Symposium on Applications and the Internet 2002 (SAINT 02).
[25] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, MIT Univ. Press, 2001.
[26] T. Srinivasan, M. Sandhya, N. Srikrishna, An E□cient Parallel IP Lookup Technique using CREW based Multiprocessor Or-ganization, Proceedings of the 4th Annual Communication Networks and Services Research Conference 2006 (CNSR06).
[27] W. Chen, C. Tsai, A Fast and Scalable IP Lookup Scheme for High Speed Networks, Proceedings of the 7th IEEE Interna-tional Conference on Networks 1999.
[28] W. Jiang, V. Prasanna, Sequence-preserving parallel IP lookup using multiple SRAM-based pipelines, Elsevier, Journal of Dis-tributed Computing 69, pages 778-789,2009.
[29] Y. Chang, Simple and fast IP lookup using binomial spanning trees, Elsevier, Journal of Computer Networks 28, pages 529-539, 2005.