Fast and Scalable IP Address Lookup with Time Complexity of $Log_mLog_m(n)$

Abhishant Prakash
Motilal Nehru National Institute of Technology, Allahabad (INDIA)
Email: prakashabhishant@gmail.com

Alok Sagar Gautam
Hemawati Nandan Bahuguna Central University, Garhwal (INDIA)
Email: phyalok@gmail.com

***Abstract*— Router is a device which is used to route the incoming data packets from the sender to the destination using the IP address of the destination side. Since the router performs the task in the real time so there are millions and millions of the data packets which the router has to process. The main task in the routing process is to find the corresponding port number for a particular IP address entry stored in the router. Since the router performs the searching task for the millions of incoming data packets an efficient searching algorithm is required to be performed on the routing table. In this paper we proposed an efficient searching algorithm for the routing table of the routers. Various algorithms has been performed in this field with the best time complexity of $log_m(n)$ of the balanced m way search tree and $log_2log_2(n)$, we in this paper proposed an efficient searching algorithm with the time complexity of $log_mlog_m(n)$ in the worst case applicable to both the IPV4 and IPV6.**

***Index Terms*— IP address lookup, IPV6, $log_mlog_m(n)$ complexity, IPV4.**

I. Introduction

The internet technology has seen an admirable change in the last decade. With the increasing number of hosts and domains over the internet the higher bandwidth and high speed data packet transfer has always been a topic attracting scientists and researchers from all over the world. With the increasing number of hosts over the internet a new Internet Protocol Version named IPV6 came into existence. The data packet transfer is mainly handled by the router who's task is to forward the incoming data packets to their destination IP address. The data packets consists of two fields namely the senders IP address and receiver's IP address. The task of the router is to forward the data packet to the correct port number using the longest matching prefix theorem.

As router performs IP address lookup for hundreds and millions of data packets these require the secondary storage devices since the primary memory does not have enough sufficient space to store all the IP address with their respective port number. The efficiency of any algorithm is based on its memory size required, lookup speed, pre-processing requirements, scalability and flexibility towards the routing table. The most important among them is the lookup speed which is directly associated with the number of memory accesses required for searching. The proposed algorithm makes use of the balanced m way search tree and the hash table to search for any prefix, search gets completed in the following two cases first, when there is a matching prefix, second when there is no match found and in case of matching the output port is remembered and returned in order to transfer the data packet to its destination address as both IP address and output port is required to transfer the data packet to its destination.

The main idea behind this paper is to make use of the software based scheme for IP address lookup which requires some microcontroller operations.

II. IP Address Lookup Problem

A. Related Work

A.1 Hardware Based Schemes

Several schemes have been proposed in this direction including both the hardware based schemes and software based schemes. The most important among them are as follows:

McAuley, Francis proposed the structure of the TCAM [1]. TCAM stands for Ternary Content Addressable Memory. It is of the same size as of the Static RAM but has higher power consumption, small size and to add more it has high cost. Figure 1 shows the basic structure of the TCAM

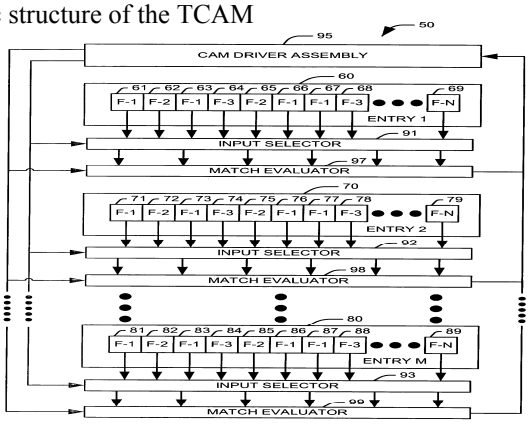


Fig 1. Basic Structure of TCAM

doi:10.4304/jait.5.2.58-64

Song and Longwood gave the extension to the TCAM, an architecture called BV-TCAM [2] which combines the TCAM and bit vector algorithm to effectively compress the data representation and boost the system throughput. Another hardware based scheme includes of cache based IP address lookup architecture proposed by Kasnavi, Berube, Gaudet, and Amaral [3], which is comprised of non-balancing Multizone Pipeline Cache (MPC) and of a hardware-supported IP routing lookup method. They use two stage pipelines for a half prefix/half full IP address cache that results in lower activity than conventional caches. . IP address lookup has also been proposed using ring pipeline architecture for tree based search engines by Baboescu , Tullsen, Rosu, and Singh [4]. The pipeline stages are configured in a circular multi point access pipeline so that search can be initiated at any stage. Two more algorithms named Elevator-stairs and log W-elevators [5] have been proposed by Sangireddy, Futamura, Aluru, and Somani, which is scalable and memory efficient. RAM based hardware architecture with the lookup speed of 66 MLPS have also been proposed in this area by Meribout and Motomura [6]. In this design a commodity Random Access Memory (RAM) is needed in their design; and achieved lookup rate is reasonably low. Another scheme by Gupta, Lin, and McKeown includes a route lookup mechanism implemented in a pipeline fashion with speed of one route with every memory access [7].

A.2 Software Based Models

Several software based model have been proposed the most common tree based model including Ruiz –Sanchex, Biersack, Dabbous model [8]. Figure 2 shows the general trie structure. Trie stores prefixes into the node and each node is defined by the path from the root node of the trie but the main disadvantage of this scheme was that a lot of empty internal nodes were present which increases the number of memory accesses. Another tree based architecture IP address lookup includes balanced binary search tree by Lim, Kim and Yim [9] which uses Binary Prefix Tree theorem to sort the different prefixes in their increasing order. In order to reduce the number of memory accesses in a binary trie multibit examining by Chao [10] and examining more than one bit at a time using prefix expansion by Srinivasan , Varghese [11] and level compressed tries applies i.e. multibit tries with path compression techniques by Nilsson, Karlsson [12].Balanced m way tree structure includes IP address lookup using balanced multiway tree by Lim, Kim and Lee [13] in which different prefixes were sorted with BPT theorem and were arranged in the form of balanced m way tree with time complexity of $\log_m n$.

A.3 Hashing Based Schemes

Several hashing based schemes have been proposed in this area. The first one is by Wadvogel and Varghese to organize a routing table by prefix length and apply binary search on the prefix length in the routing table. One of the most important proposed schemes in this area is of lulea scheme by Yu, Mahapatra, Bhuyan [15]. The scheme reduces table into small data structures which fits into the cache. But the major disadvantage of this scheme is the incremental update since it requires a complex preprocessing and also it cannot be used for larger table.

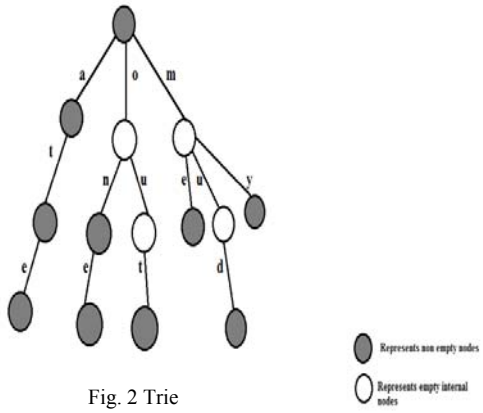


Fig. 2 Trie

III. PROPOSED SCHEME

A. *Prefix Sorting.*

The following table I shows the set of prefixes for which the proposed algorithms is performed. The prefix taken makes utilization of the first three bits of them to identify the family of prefixes. Similarly the algorithm can be implemented to find the network part of the incoming data packets from the first four bits of the destination IP address. The proposed algorithm makes utilization of the BPT theorem in order to sort the prefixes in their sorted order.

TABLE I:

PREFIX TABLE FOR PROPOSED ALGORITHM

Prefix Name	Binary Equivalent
a	(000)
b	(001)
c	(111)
d	(100)
e	(111)01
f	(100)111
g	(010)
h	(111)11
i	(011)
j	(110)
k	(010)011
l	(110)101
m	(101)
n	(000)11
o	(000)101
p	(001)011
q	(000)001
r	(000)111
s	(001)11
t	(000)01
u	(001)000
v	(101)01
w	(001)001
x	(011)110
y	(101)11
z	(100)10

The set of prefixes taken were arranged into different sets of balanced m way tree. In the proposed algorithm the first three bits of the prefixes are used to build the

hash the hash table and each hash table has a pointer to their respective balanced m way tree. The prefixes in the m way tree are stored with the BPT theorem which states that for two prefixes P_i and P_j with $n(P_i) < n(P_j)$ and $S(P_j, n(P_i))$ defines the most significant bits of the prefix P_j then P_i is smaller than the prefix P_j if $P_i < S(P_j, n(P_i))$ and else vice versa. The third condition which arises is case of equality in that case the $(n(P_i)+1)$ th bit of P_j is checked if it's equals to 1 then Pj is greater else P_i is greater.

The main idea behind this approach is to build the balanced m way tree in different segments.

to build the hash table, so the size of the hash table is 2^3 similarly to identify the network from the IP address of the receiver IP address from the data packet first 4 bits can be used to build the hash table.

The hash table contains the two fields: first the pointer to its sub tree and second the key part which is either zero or one. Zero entry in the hash table defines no sub tree whereas the entry one denotes presence of the sub tree in the respective field of the hash table

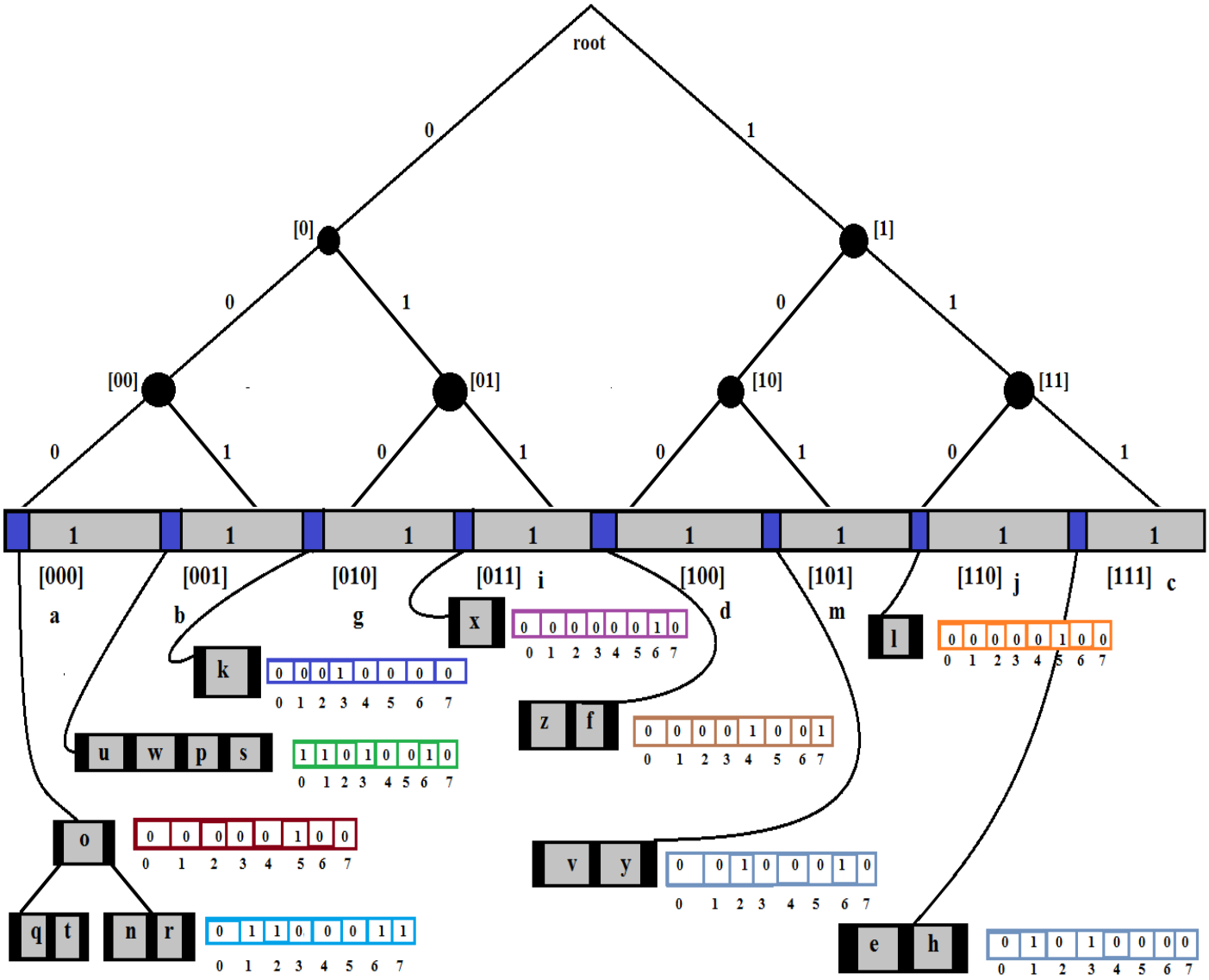


Fig 3. Proposed Tree Structure for the Algorithm.

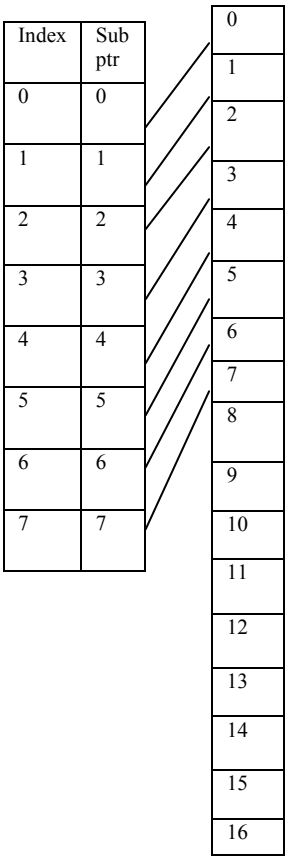
B. *Constructing the Tree*

The proposed tree consists of two segments; first segment consists of the hash table. In our proposed algorithm we have opted for first three bits of the prefixes

The second segment of the tree consists of the different segments of balanced m way tree. Fig. 3 shows the tree structure of the proposed algorithm. For the given fig. 3 we have made balanced 5 way tree which can be generalized for m way tree. At each level of the m way

tree we have defined the hash table so that search can be performed faster. Since we have taken prefixes of maximum length of six and there are prefixes starting with a particular three bit segments in a particular sub segment of the main hash table so the size of the hash table at each sub tree level is 2^3. Hence by making the hash table we have reduced the time complexity for the searching purpose.

TABLE II
CONSTRUCTED ROUTING TABLE FOR PROPOSED SCHEME



A	B	C	D	E	F	G	H	I	J	K	L	M
0	a(000)	8	0	-	-	0	-	-	0	-	-	0
0	b(001)	10	0	-	-	0	-	-	0	-	-	0
0	g(010)	11	0	-	-	0	-	-	0	-	-	0
0	i(011)	12	0	-	-	0	-	-	0	-	-	0
0	d(100)	13	0	-	-	0	-	-	0	-	-	0
0	m(101)	14	0	-	-	0	-	-	0	-	-	0
0	j(110)	15	0	-	-	0	-	-	0	-	-	0
0	c(111)	16	0	-	-	0	-	-	0	-	-	0
1	o(000)101	-	1	-	-	0	-	-	0	-	-	0
0	q(000)001	-	0	t(000)01	-	0	n(000)11	-	0	r(000)111	-	0
0	u(001)000	-	0	w(001)001	-	0	p(001)011	-	0	s(000)11	-	0
0	k(010)01	-	0	-	-	0	-	-	0	-	-	0
0	x(011)10	-	0	-	-	0	-	-	0	-	-	0
0	z(100)10	-	0	f(100)111	-	0	-	-	0	-	-	0
0	v(101)01	-	0	g(101)11	-	0	-	-	0	-	-	0
0	l(110)101	-	0	-	-	0	-	-	0	-	-	0
0	e(111)01	-	0	f(100)111	-	0	-	-	0	-	-	0

A: Child Valid 1 B. Prefix 1 C. Subtree Pointer 1 D: Child Valid E. Prefix 2 F: Subtree Ptr. 2
G: Child Valid 3 H: Prefix 3 I: Sub tree ptr. 3 J: Child Valid 4 K: Prefix L: Sub tree ptr. 4

C. Constructing the Routing Table

The routing table is constructed with the two segments in it. The first segment is the range table which contains the entry from 0 to 2^3-1. To efficiently perform the searching we have constructed the range table and apart from this there is a main table which contains various entries. Table II shows the constructed routing table.

The Table II shows the routing table with the range table and range table and the main table. To look for any prefix first search is done on the range table and then it will be done on main table.

D. Constructing the Routing Table

The search gets performed in the following two cases when there is a matching prefix found or when there is no matching prefix found. In both the cases the output port is remembered and is returned after the search gets completed. For e.g. for the input of 000001110001 the first three are used for the searching purpose in the main hash table. The search is redirected to a's sub tree at each level search is performed in the hash table first and then search is carried in the fashion of m way search at its child nodes. The search gets completed at the prefix q and the longest matching prefix is 000001 and hence output port corresponding to the q prefix is returned.

E. Updation.

The algorithms are considered on the fact of insertion and deletion also. To insert any element if there is enough space in the routing table as well we just need to find the main hash table entry where it belongs to and the prefix can be inserted into the m way tree and correspondingly hash table entry can be modified and similar modification can be done in the routing table. In the case of deletion the node can be deleted from the m way tree of the

respective main hash table and in same manner it can be deleted from the routing table.

IV. Performance Analysis

A. *Complexity Analysis.*

In the proposed algorithm we first find the main hash table entry and if any sub tree is present in it we proceed to it. Searching in the main hash table takes O(1) time. In the search down the balanced m way tree, at each level we look into the corresponding hash table for the search node if its present search gets completed there and output port is returned in case it's not matched the search continues in the further depth. The proposed algorithm works in $\log_m \log_m(n)$. To prove the time complexity first we find out the height of the balanced m way tree. Fig. 4 shows the general structure of m way tree .From this we find out the height of balanced tree.

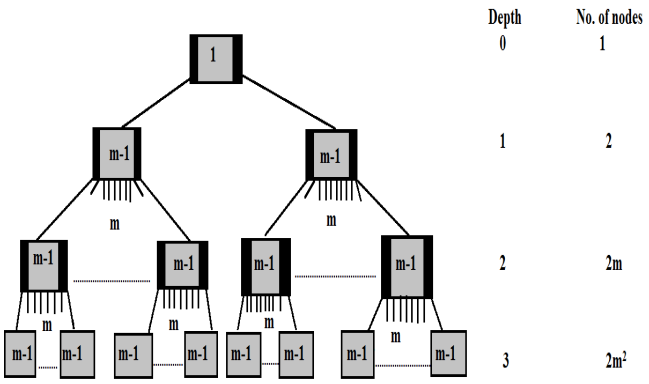


Fig 4. General structure of m way tree.

The number of keys:

$$n >= 1+ (m-1) \sum_{i=1}^{i=h} (2m^{m-1}) \quad (1)$$

$$= 1+2(m-1) \{(m^h - 1)/ (m-1)\} \quad (2)$$

$$= 2m^h -1 \quad (3)$$

Therefore,

$$m^h <= (n+1)/2 \quad (4)$$

Taking $\log_m$ on both the sides we get:

$$h <= \log_m(n+1)/2 \quad (5)$$

So the height of the balanced m way tree is $\log_m(n)$. Now at each level the m way search is performed in the proposed algorithm and looking into the hash table takes O(1) time so the total complexity of the proposed algorithm comes out to be $\log_m \log_m(n) + O(1)$. Hence the proposed scheme gives the time complexity of $\log_m \log_m(n)$. In the worst case the m way tree splits in m/2 ways so the worst case height of the tree structure would $\log_{m/2}(n)$, hence in the worst case the time complexity of the proposed algorithm is $\log_m \log_{m/2}(n)$.

B. *Performance Evaluation.*

The following table III shows the performance evaluation of the proposed algorithm and various previous algorithm for various input size and their corresponding time complexity.

TABLE III:
PERFORMANCE ANALYSIS

Input Size (N)	Binary search	LC-Trie	M-way tree(m=5)	Proposed Scheme
	$Log_2(N)$	$Nlog_2(N)$	$Log_m(N)$	$Log_m log_m(N)$
4.0000	2.0000	2.0000	0.8674	-0.0920
16.0000	4.0000	64.0000	1.7228	0.3379
1000.0000	9.9657	9965.7842	4.2920	0.9051
10000.0000	13.2927	132877.1238	5.7227	1.0838
100000.0000	16.6096	1660964.064	7.1534	1.2225

The following fig. 5 shows the graphical analysis of the proposed scheme and previous scheme for the set of results given in Table III.

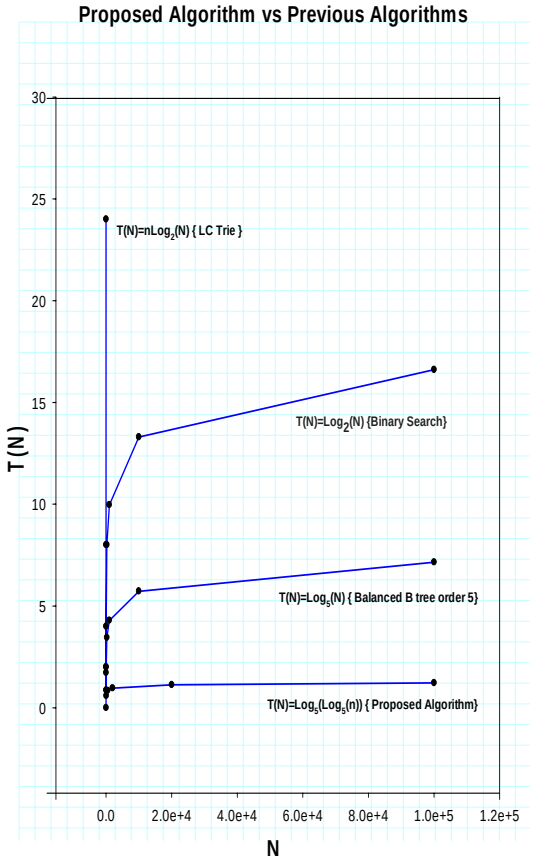


Fig.5 Graphical Analysis

C. Pseudo Code for Searching

```
Search_prefix (prefix p, struct node *n, int  *pn)
{
  //look corresponding entry in hash table at the level
                          //log(n)/2
 hash[log(p)/2]==1 then
 call search( hash[log(p)/2],prefix p, int* pn);
 else
 remember the output port;
 return;
 else
 call search(prefix p, n->child(*pn),pn);
 }
 Search (struct node * n, prefix p,  int* pn)
 {
 If p==NULL
 return;
 Search for the next three bits value label in hash table
 if(entry in hash table==1) then
 remember output port;
 if (p<n->key[1])
 {
 *pn=0;
 return0;
 }
 Check for N bits of prefix with matched prefix
 if (equal)
 return current remember output port;
 else return previously remembered output port;
 else
 *pn=n->count //total keys in current node
 While(prefix<n->key(*pn)&&*pn>1)
 {
 (*pn)--;
 }
 return 0;
 }
```

V. CONCLUSION.

The paper presented the IP address lookup for the routers with time complexity of $\log_m \log_m(n)$ where n is the total no. of bits of the prefix. The tradeoff which can be used for this scheme to build first hash table is $\log_2(n)/2$. The proposed scheme is compatible for both the IPV4 and IPV6 as the proposed scheme depends upon the number of prefixes and not on the length of the prefixes. As far as Updation is concerned the proposed scheme is partially updatable which depends upon the number of left entries in the routing table. If there are vacant entries in the routing table then Updation is possible.

VI. REFERENCES

[1] A. McAuley, P. Francis “Fast routing lookup using CAMs,” Proc. IEEE INFOCOM 93(1993) 1382-1391.

[2] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," in Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays Monterey, Cali fornia, USA: ACM, 2005.

[3] S. Kasnavi, P. Berube, V. Gaudet, and J. N. Amaral, "A cache-based internet protocol address lookup architecture," Comput. Networks, vol. 52, no. 2, pp. 303-326, 2008

[4] 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, 2005.

[5] R. Sangireddy, N. Futamura, S. Aluru, and A. K. Somani, "Scalable, memory efficient, high-speed IP lookup algorithms," IEEE/ACM Trans. Netw., vol. 13, no. 4, pp. 802-812, 2005.

[6] M. Meribout and M. Motomura, "A new hard- ware algorithm for fast IP routing targeting pro- grammable routers," in Network control and engineering for Qos, security and mobility II: Kluwer Academic Publishers, 2003, pp. 164-179.

[7] P. Gupta, S. Lin, and N. McKeown, “Routing lookups in hardware at memory access speeds,” in Proc. IEEE INFOCOM’98, Session 10B-1, San Francisco, CA, pp. 1240–1247.

[8] M.A. Ruiz –Sanchex, E.W. Biersack, W. Dabbous,” Survey and taxonomy of IP address lookup algorithms”, IEEE networks (2001) 8-23.

[9] H. Lim, H. Kim, C. Yim,” IP address lookup for internet routers using balanced binary search trees,” IEEE transactions on Communications Vol. 57, no.3 March 2009.

[10] H. Jonathan Chao,” Next generations routers,” Proc. IEEE 90(9) (2002) 1518-1558.

[11] V. Srinivasan , G. Varghese,” Fast Address lookup using controlled prefix expansion,” in Proc. IEEE ACM sigmetrics 98 conf., Madison WI, 1998 pp. 1-11.

[12] S. Nilsson, G. Karlsson,”IP address lookup using LC tries,” Proc. IEEE J.Select.areas Commun. 17(1999) 1083-1092.

[13] H. Lim, W. Kim, B. Lee, C. Yim,” High-speed IP address lookup using balanced multiway tree,” Computer Communications 29(2006) 1927-1935.

[14] M. Wadvogel, G. Varghese, J. Turner, B. Plattner,” Scalable high speed IP routing lookups” in Proc. IEEE ACM SIGCOMM 97 Cannes, France 1997 pp. 25-35.

[15] H. Yu, R. Mahapatra, L. Bhuyan,” A Hash-based Scalable IP lookup using Bloom and Fingerprint Filters,” in Proc. Global Comm. Conference IEEE(2009).

Abhishant Prakash is an undergraduate student in the National Institute of Technology, Allahabad India. His area of interests includes networking & communication and algorithms design. He has attended various National Conferences and has presented papers in them.

Alok Sagar Gautam is working as Assistant Professor in the department of physics, HNBGU Central University Garhwal, Uttrakhand, India. Since 2006, he was scientist at Indian Institute of Tropical and Meteorology, Pune and during his employment at Pune, He participated in various observational

field campaigns i.e. 28th Indian Scientific Expedition to Antarctica, STORM CAIPEEX program and published several scientific papers in good impact factor journals. He has also Junior Associate of ICTP Italy. He participated several National and International conferences and presented the papers and published the papers in proceedings.