INTRODUCTION

Forwarding of Internet Protocol (IP) packets is the primary purpose of Internet routers. Packet Classification is an important task performed by routers to classify packets i.e. to send a packet to the next hop. The internet is a collection of interconnected routers. The communication between them takes place using internet protocol known as IP. This IP datagrams (also known as "Packets") travel over one link to another and from its source to its destination via a number of routers. The packet classifier does the job to select the next hop for an incoming packet based on some filters. Longest Prefix Matching (LPM) is an integrated part of Packet Classification it is done to determine the longest matched prefix with an incoming address. Here, the address means the destination IP address that can be extracted from the packet header. An IPv4 packet header consists of five tuples of 104 bits length; the tuples are namely Source IP Address (SIP), Destination IP Address (DIP), Source Port (SP), Destination Port (DP) and Protocol (8 Bits). In IPv4 format the IP addresses are of length 32 bits, so, the SIP and DIP are of 32 bits length. The SP and DP fields are of length 16 bits and the protocol field is of 8 bits length, these totals to 104 bits. The value of protocol field can either be Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

Prefix means the fixed part of an IP address expressed using Classless Interdomain Routing (CIDR) notation. Let us take an example of IP address represented in CIDR notation—192.168.100.0/24 is an IP address expressed in CIDR notation, here /24 means the number of fixed bits i.e. 24 bits are fixed for this IP address and rest 8 bits can vary, each of them can either be 0 or as a result this address covers a range of addresses starting from 192.168.100.0 to 192.168.100.255.

Source IP Address (32 bits)	Destination IP Address (32 bits)	Source Port (16 bits)	Destination Port (16 bits)	Protocol (8 bits)

Fig. 1: Format of IPv4 Packet Header

IP address lookup is the process to find out an output port and a next hop address corresponding to the destination IP address of each incoming packet. It is a basic operation for packet forwarding in Internet routers. The IP address has a hierarchical structure of a network identifier and a host identifier. The network identifier is termed a prefix. The IP address lookup is solely based on prefix; it searches for a prefix that matches the destination IP address of each incoming packet from the pre-defined routing table.

Longest prefix match (also called Maximum prefix length match) refers to an algorithm used by routers in Internet Protocol (IP) networking to select an entry from a forwarding table.

Because each entry in a forwarding table may specify a sub-network, one destination address may match more than one forwarding table entry. The most specific of the matching table entries — the one with the longest subnet mask — is called the longest prefix match. It is called this because it is also the entry where the largest number of leading address bits of the destination address match those in the table entry.

For example, consider this IPV4 forwarding table (CIDR notation is used):

192.168.20.16/28
192.168.0.0/16

Fig. 2: IPV4 forwarding table

When the address 192.168.20.19 needs to be looked up, both entries in the forwarding table "match". That is, both entries contain the looked up address. In this case, the longest prefix of the candidate routes is 192.168.20.16/28, since its subnet mask (/28) is longer than the other entry's mask (/16), making the route more specific.

Forwarding tables often contain a default route, which has the shortest possible prefix match, to fall back on in case matches with all other entries fail.

CHAPTER 1: Problem Details

1.1 Problem Statement

Longest Prefix Matching using Hash Functions.

1.2 Problem Definition

Here we are trying to propose a hash based solution for Longest Prefix Matching for Packet Classification. We are trying to design a hash function which will minimize the number of searches required for matching the longest prefix. Now the major problem which we are facing in this approach is collision which we want to minimize. So basically we want to minimize the search time with as less number of collisions as possible.

1.3 Problem Objective

The main objective of designing this method is to eradicate the problems that we faced from the following list of methods:

- Trie based solution faces the problem that every time we need to make any update to the existing trie we would have to reconstruct it again making the solution slower. So we are focusing on methods where making any changes should not cause any extra overhead.

- In TCAM based approach, prefixes are needed to be sorted in descending order of their length which leads low throughput. So we needed to look for such a solution which overcomes this issue too.

- In some of the TCAM based methods where sorting is not needed another major issue is high power consumption and high cost. In order to reduce cost BiCAM, TCAM pair is used for storing prefixes.

1.3 Tools and Platform

Programming Language Used	Verilog , C++
Compiler Used	ModelSim, Dev C
Operating System	Windows 2000 or above
Primary Memory (Minimum)	512 MB RAM
Disk Space Required	Program files require approximately 750MByte of disk space

1.4 Brief Discussion on Problem

Packet Classification involves various types of matching, e.g. **L**ongest **P**refix **M**atching (LPM), exact matching, and range matching. The procedure is to extract the destination IP address from the packet header and then pass on to the LPM processor, the output of that is passed to the range matching processor and exact matching processor. If the range matching test is successful then the packet is delivered via that destination port.

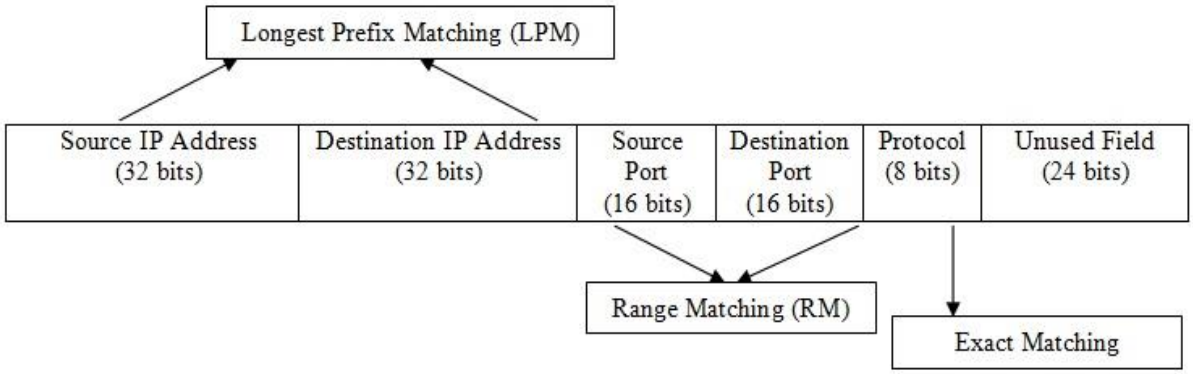


Fig. 1.1: Process of Packet Classification on IPv4

A) Range Matching

Range matching is a challenging problem in packet classification due to the problems associated with storing ranges. Ranges need to be stored in an efficient way to simplify the matching of the header of an incoming packet with the rules stored. Hence, efficient storage of ranges and high speed range matching is needed. The main problem in maintaining rule database is the presence of ranges in the Source Port and Destination Port fields due to requirement of multiple entries for storing just a single rule in the rule table. In order to improve the storage efficiency of the rule table in a memory, several approaches, both hardware and algorithmic have been put forward. The range

of source port and destination port is also present in a rule table so it needed to be checked that whether the source and destination port selected by a LPM processor is within the range or not.

B) Exact Matching

Exact matching is done only with the protocol field. The value of this field is mainly **T**ransmission **C**ontrol **P**rotocol (TCP) or **U**ser **D**atagram **P**rotocol (UDP). After receiving an IP packet the value of protocol field is extracted and then that value is matched with the value of protocol field of an incoming IP packet. This field need to be matched exactly.

C) Longest Prefix Matching (LPM)

Each entry in an IP forwarding table consists of a prefix and its associated next hop information. A prefix may be of an arbitrary length, and it consists of a bit string specifying the initial substring of a network address. No two entries contain the same prefix. An LPM search is a search technique to find out the longest prefix among the matched prefixes for a specific IP address.

Prefix	**Next Hop**
10*	**7**
01*	5
110*	3
1011*	**5**
0001*	0
01011*	7
00010*	1
001100*	2
1011001*	**3**
101100*	5
0100110*	6

Table 1.1: Sample Forwarding Table shows the LPM for an incoming destination IP address 101100101000 (sake of simplicity 12 bits shown), shows matches in 4^{th}& 9^{th} location (counting starts from top).

The main problem with LPM is that it is 2D search i.e. the value of a prefix as well as the length of the prefixes needs to be checked. In Table 2.1 the incoming address is 101100101000 and a part of it matches with 3 entries (i.e. 10*, 1011*, 1011001*).The length of first matched prefix is 2 bits as the first two MSB's of the incoming address is matched with the prefix and the rest of the bits of the

stored prefix is don't care. Similarly, for the stored prefix 1011* the length is 4 and for the last one it is maximum. So, among those three prefixes, since the last prefix (1011001*) matches with the maximum number of bits of the incoming string (101100101000), therefore, LPM will select 1011001* as its output.

There are many existing software and hardware approaches for performing Longest Prefix Matching those are explained in details in the following sub-sections.

a) Trie based approaches

In computer science, a trie, also called digital tree and sometimes radix tree or prefix tree, is an ordered tree data structure that is used to store a dynamic set or associative array. The difference between tree and trie is that in case of a tree the data is stored as a part of the node but in a trie the data is stored along with a path. If a trie is traversed from the root to leaf node then the path from root to leaf will be a string. In the trie based LPM solutions the prefixes are represented in a prefix tree and the tree is traversed according to a given search key. They mainly follow Depth First Search technique.

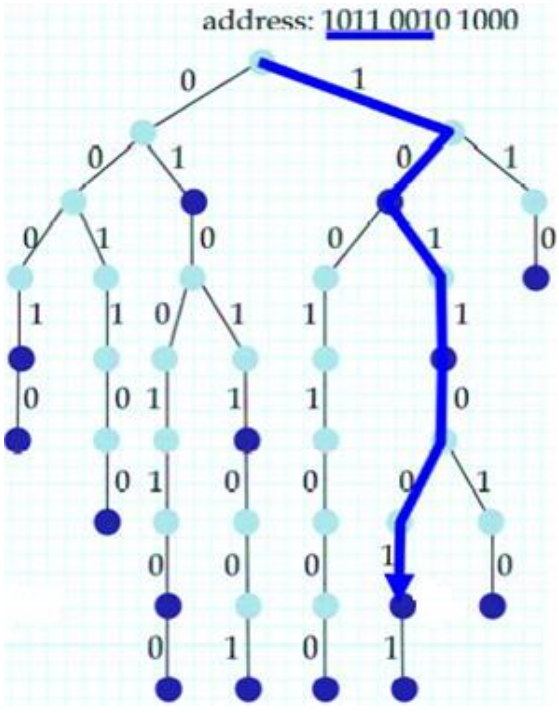


Fig.1.2: Example of a 1-bit Trie for Table 1.1

There are single bit and multi bit tries but the main problem of trie based approaches is that it requires a lot of memory and as a result it is slower. Figure 1.2 is an example how a 1-bit trie is made and how the prefixes are matched, here the trie consists the prefixes of Table 1.1.

b) Ternary CAM based hardware Solution

A Content Addressable Memory (CAM) is able to search itself in a very short time as the comparison is done in parallel with a search key. In addition to data bit string, a mask bit string that indicates those bit positions for which the content bit string does not need match the given key. Fig 2 illustrates the searching process in the existing TCAM based approaches. Prefixes are stored in descending order of their lengths. When update of forwarding table comes the entire TCAM needs to be sorted that is time consuming. When a search is performed then the search key is compared in parallel with all the entries of TCAM and the match vector is sent to the priority encoder and the output of that is the required result.

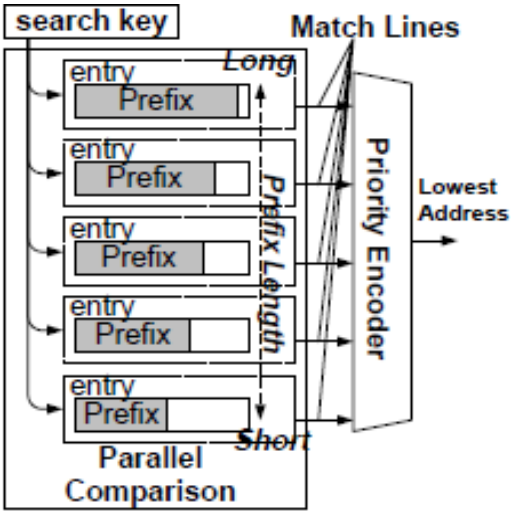


Fig.1.3: Existing TCAM based LPM solution

1.4.1 Some key definitions

- **IP address:**

An **Internet Protocol address** is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network that uses the Internet Protocol for communication.

- **IP prefix:**

IP prefix is a prefix of IP address. All computers on one network have same IP prefix. For example, in 192.24.0.0/18, 18 is length of prefix and prefix is first 18 bits of the address.

- **IP datagram:**

The format of data that can be recognized by **IP** is called an **IP datagram**.

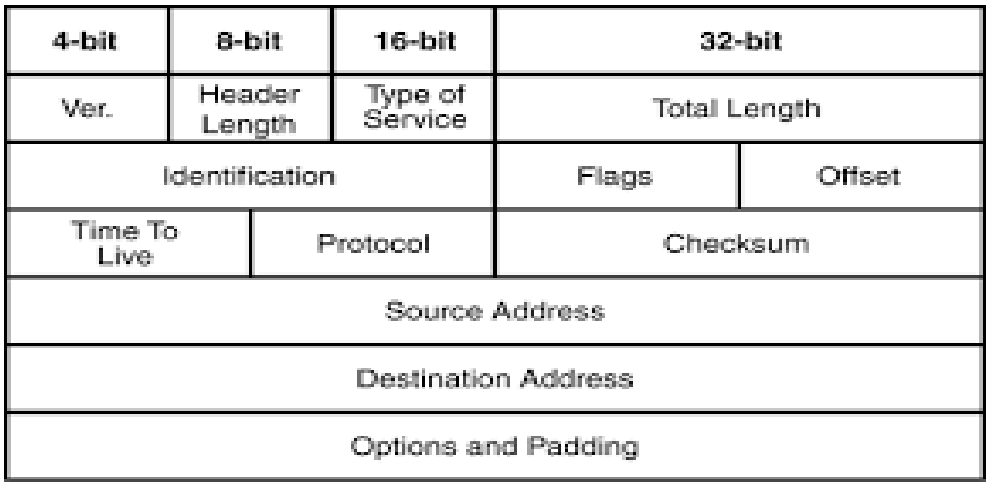


Fig. 1.4: IP Datagram

- **Router:**

A router is a networking device that forwards data packets between computer networks. Router perform the traffic directing functions on the Internet. A data packet is typically forwarded from one router to another through the networks that constitute the internetwork until it reaches its destination node.

- **Routing table:**

A routing table uses a packet's destination IP address to determine which IP address should next receive the packet, that is, the "next hop" IP address.

- **Forwarding table**:

A forwarding table uses the "next hop" IP address to determine which interface should deliver the packet to that next hop, and which layer 2 address (e.g., MAC address) should receive the packet on multipoint interfaces like Ethernet or Wi-Fi.

- **Classful Addressing:**

The method divides the address space for Internet Protocol Version 4 (IPv4) into five address classes by address range. Classes A, B, C are networks of three different network sizes, i.e. number of hosts for unicast addresses.

- **Next hop**:

Next hop is a routing term that refers to the next closest router a packet can go through.

- **CIDR over classful addressing:**

Originally, IP addresses were assigned in four major address classes, A through D. Each of these classes allocates one portion of the 32-bit IP address format to identify a network gateway -- the first 8 bits for class A, the first 16 for class B, and the first 24 for class C. The remainder identify hosts on that network -- more than 16 million in class A, 65,535 in class B and 254 in class C. (Class D addresses identify multicast domains.)

CIDR reduced the problem of wasted address space by providing variable length network id.

Chapter 2: Concepts and Problem Analysis

2.1 Concept of Longest Prefix Matching:

Since prefixes might overlap (this is possible as classless addressing is used everywhere), an incoming IP's prefix may match multiple IP entries in table.

For example, consider the following table:

Prefix	Next Hop
192.24.0.0/18	D
192.24.12.0/22	B

Fig. 2.1: Next Hop table

In above table, addresses from 192.24.12.0 to 192.24.15.255 overlap, i.e., match with both entries of the table. To handle above situation, routers use Longest Prefix Matching rule. The rule is to find the entry in table which has the longest prefix matching with incoming packet's destination IP, and forwarding the packets to corresponding next hope.

In the above example, all packets in overlapping range (192.24.12.0 to 192.24.15.255) are forwarded to next hop B as B has longer prefix (22 bits).Therefore longest prefix matching is a necessary task in Classless Inter Domain Routing to forward data packet to its destination. But as we have seen there are some problems in previous specified algorithms in respect of time and space complexity in searching the prefix. The software based trie structure has problem with height of trie and need of rebuilding the trie for minute changes. Hardware based approaches suffer cost and register issues as more the number of registers more the cost and there could be heating problems as well.

The Longest Prefix Match is refers to an algorithm to match the longest length of prefix from the forwarding table. Since one destination address can be match with one or more prefix in the table.

In CIDR format it may happen that many supernetting is done to shorten the forwarding table but of those there may be only 1 group which does not belong to the same provider. For that individual group the table entry must be updated. Thus the concept of Longest Prefix Matching came into picture.

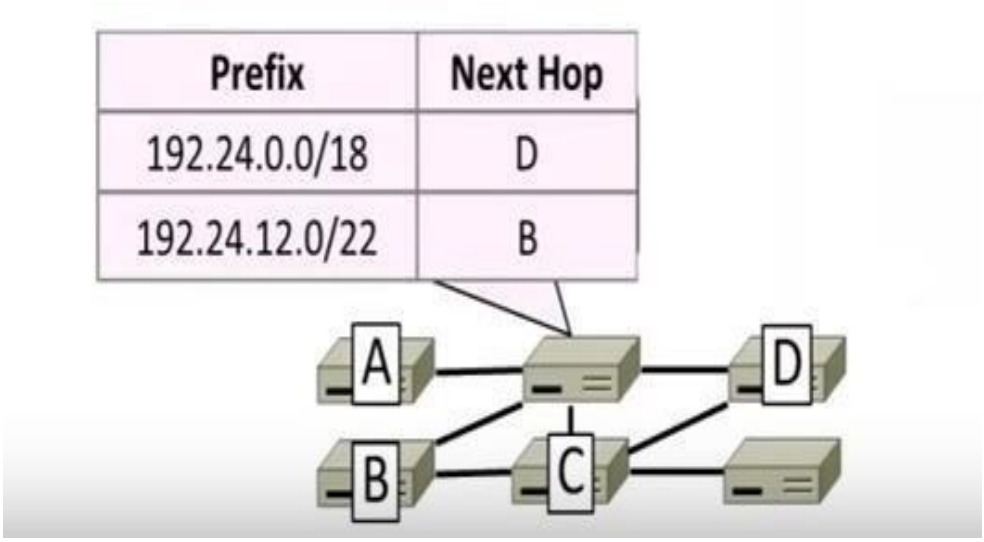


Fig. 2.2: Forwarding table

Let us suppose a router receive a packet. So it will look to its forwarding table and forward it to the next router. But the problem is that it may match to 1 or more prefix in the forwarding table.

So in that case choose the next hop with maximum length matching prefix. In the Internet routers do the routing operations. It's their responsibility to send the packet in right next hop.

For example, if the router had received two packet, one with IP **192.24.6.0** and one with **192.24.14.32**.

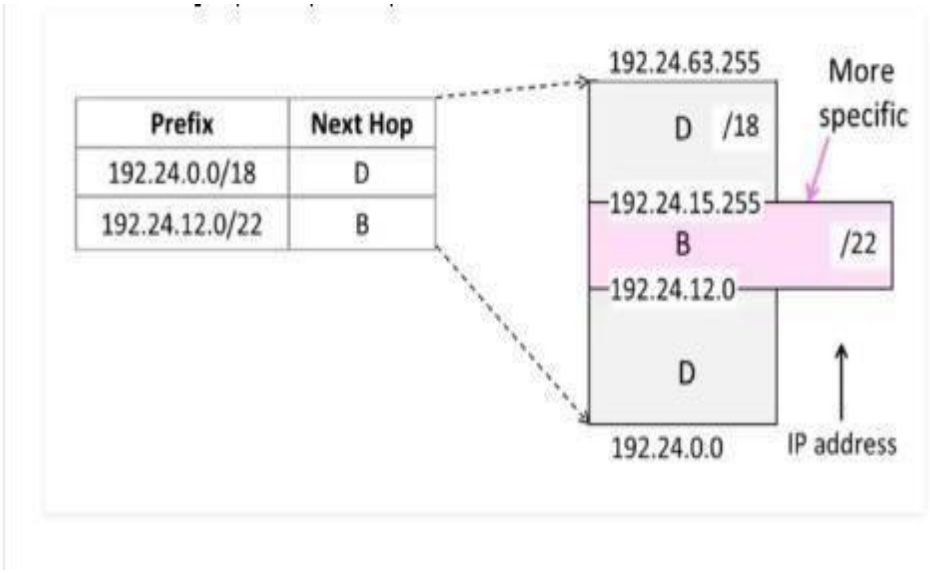


Fig. 2.3: Matching Longest Prefix

According to the forwarding table it will forward the first packet to the router **D** and the second packet to **B.**

Hierarchical Structure for LPM:

In order to meet the requirement of high-speed IP processing, a high-throughput low-power lookup table architecture is requisite. This novel architecture provides an efficient and extremely fast hardware structure for LPM, which involves basic gates and a hierarchical structure of forwarding table to reduce power consumption and to increase the efficiency of LPM technique. This hierarchical structure of forwarding table does not arrange prefixes according to their prefix lengths

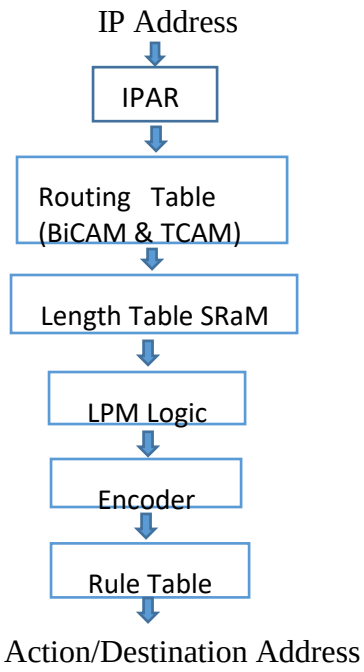


Fig. 2.4: Block diagram of IP processing for LPM

For each incoming packet to a router, at first the destination IP Address of the packet is retrieved and stored in a register named IP Address Register (IPAR). In order to identify the next hop address of an incoming packet, the address in the IPAR, needs to look into a table called routing table. Since the table employs CAM for storing prefixes, so the IP address present in the IPAR is compared with a set of entries stored in a routing table with a search time complexity of O(1) due to parallel searching property. After the search operation in the routing table, it produces a complete comparison result indicating where the matches are found in the table.

In the proposed hierarchical architecture, the prefixes are not stored based on their length in the routing table, and therefore, to identify the longest prefix amongst all the matched ones is difficult. The identification of the longest prefix for a specific IP address is made in the LPM logic.

Chapter 3: Literature Survey

3.1 Trie

Trie is nothing but like tree data structure. Instead of having different traversal algorithm like BFS or DFS we traverse the trie in accordance to the input search prefix we get[1]. Trie where each node can have at-most two child is called binary trie. Now height of the trie can be a problem. We are using IPv4 where prefix length can be 31 at-most. But as we move to IPv6 length will increase to 127.Hence data entry on updation of will cause an overhead. Moreover every minute change in trie leads to re-construction of trie[1].An example is given below:

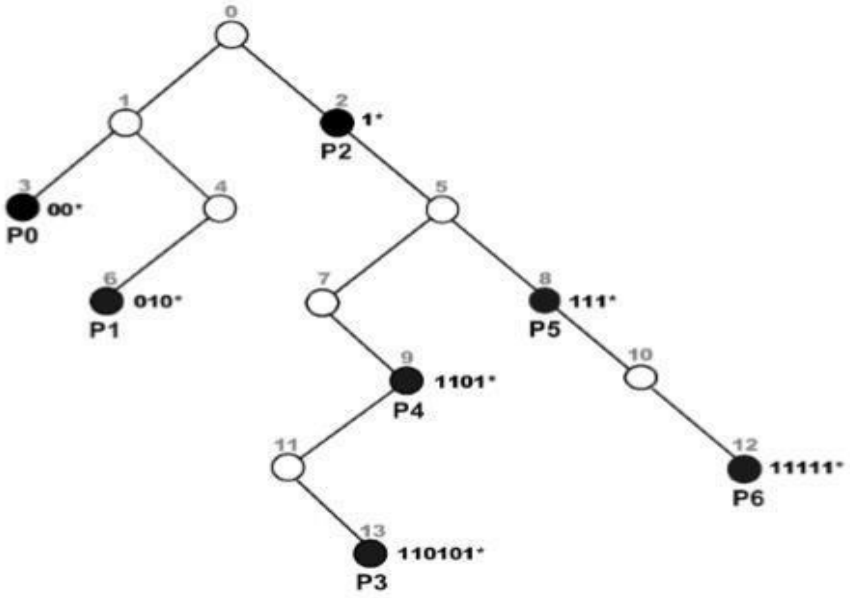


Fig. 3.1.1: Trie structure

Now more the height of the trie more exact is the match. Suppose we have input prefix as 1110. So a conflict arise as there is a match at P2 and P5. But as P5 depth is more it will be chosen.

3.2 Multibit Trie

In order to reduce length of the trie we came up with multibit trie. We took some bits called stride and according to which we had different comparisons at each node [3].An example is given below:

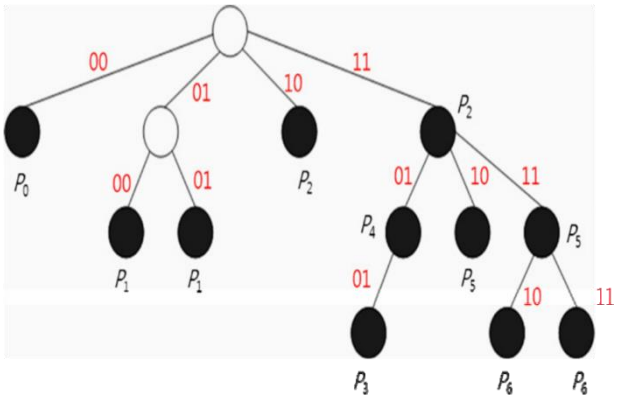


Fig. 3.1.2: Multi bit trie

Here is an example of multibit trie with stride length 2.Now we can have 4 possible numbers of comparisons with 2 bits at each node we have to compare for incoming prefixes. This sure reduces the length of the trie but those comparisons are another overhead. These Multibit trie help to reduce the height of trie but on that cost it also has a major disadvantage. The storage capacity of this structure is reduced to half. Where we can store 8 data with 8 bits of prefix with length 1 and 2. It stores only 4 prefix each of stride length 2. So when density of prefix are very high it performance is really awful. As if an odd length prefix come we have to store it adding an extra bit and if suppose a prefix came for storing where we have kept the former we are bound to store the prefixes in other location. This problem can create a chaos.

3.2 TBM

TBM is called tree bit map. Now here the concept is same as trie. Here also we take some stride length and create a super node. We do this to implement concept of EBM and IBM [6].EBM stands for External Bit Map and IBM is Internal Bit Map. EBM is used to check if any super node is present after the leaf node, and IBM is used to tell where prefixes are stored inside the super node. An example is given below:

Suppose we have EBM value as 00000100.The following structure demonstrates the structure:

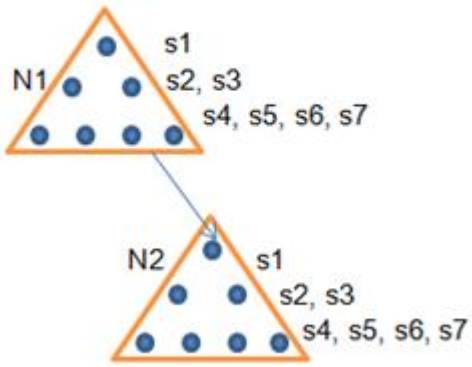


Fig. 3.2.1: TBM example

This structure uses EBM for traversal and IBM for forwarding data. Sometimes its path can be misguided. If suppose the EBM is making pointer going forward but it doesn't get a match there, so what it does is it take the IBM and forward the packet to the last present address in the IBM of that node. This can result in extra-long route since it is not guaranteed that the route which we have taken in short.

The Above Structure is showing two nodes. The first is **N1**which has internal nodes as Sn. The IBM for the **N1** is 0000000 and the EBM is 00000100. The Second Super-Node **N2** has its IBM and EBM the EBM and IBM values are as follows, 00000000 and 0000000.

Chapter 4: Design Methodology

4.1 Hardware based implementation

As said earlier we have designed a software based technique and partially done the hardware part. The basic idea behind hardware based approach is given below:

In order to reduce power consumption in this architecture, the routing table is decomposed into four sub routing tables based on the bit values of the two most significant bits (MSBs)[4], which could be '00','01', '10' and '11'. Hence four sub routing tables are formed namely SRT0, SRT1, SRT2 and SRT3.

However, only one sub table amongst four is active at a time and eventually saves power in two folds due to reduction in total number of TCAM cells present in a routing table and reduced number of active cells at a time. So, instead of storing 32 bits prefixes, only information regarding 30 bits stored in the routing tables. Each prefix is then divided into two fields namely fixed length (FL) part of 14 bits (wb) and variable length (VL) part of 16 bits (wt)[4].

In a sub table, all the FLs are stored in BiCAM while VLs are stored in TCAM and both BiCAM and TCAM are associated with their own match vector (MV) table.

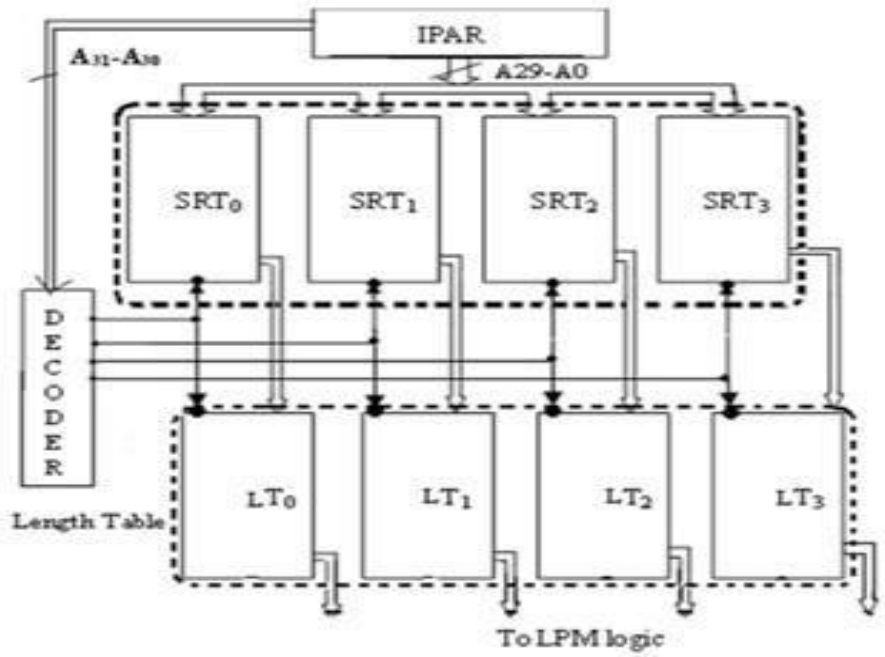


Fig. 4.1.1: Structure of hierarchical routing table

The contents of two MVs are bit wise logically AND-ed and used to maintain the final match vector (FMV) for a specific search in a sub routing table.

In reality, it is observed that nearly 99% of the prefix length varies in between 16 to 32 bit group and in IPv4.Therefore, in the proposed architecture, the length of a prefix is at least 16 bits or more, and hence, only the length information of the variable part is kept in the

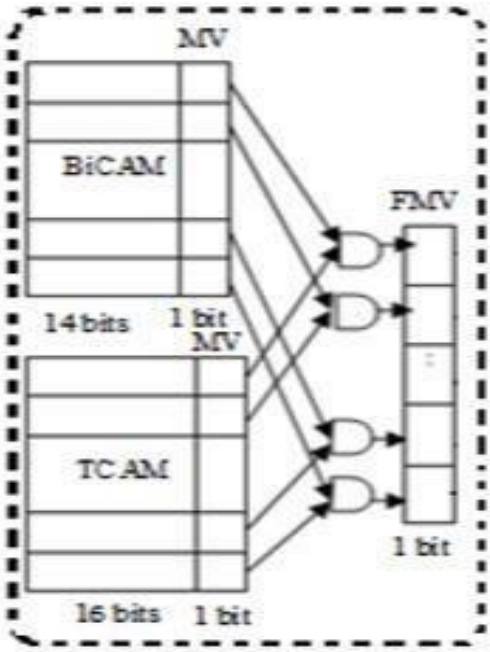


Fig. 4.1.2: Internal structure of a SRT

respective length table (LT) to maintain prefix length. However, to determine the actual length of a prefix in a sub routing table, one needs to add fixed bias value i.e. 16 with its corresponding length information in the LT. Since, the contents of LT lie in the range of 0 to 16, so, only 5 bits (log216) are used to maintain.

The architecture avoids the necessity of arranging all the prefixes according to their lengths in the routing table unlike conventional TCAM based solutions and it also provides simple and fast updates. After searching for all the matched prefixes to select the longest prefix from a pool of matched prefixes of variable lengths, a hardware structure is implemented. LPM structure consists of five identical stages where stages are selected one after another and work in a pipelined fashion. In order to determine the longest prefix from all the matched prefixes, the proposed LPM structure takes help from FMV and LT. At first, all the AND gates of the first stage receive one input from the LT (MSB:

D4) and another input from FMV, then the output of all these AND gates are logically OR-ed and used as a selection line (control) for the respective switches in that stage to pass information to the next stage. Each switch receives two inputs, one input is forwarded from the input of the AND gate and another from the output of the AND gate of that stage. In the next stage, all the AND gates receive one of the inputs from the switches and another from the LT (D3), then again the output of all these AND gates are logically OR-ed and used as a selection line for the switches in the second stage. In this stage also, all the switches receive two inputs, one from the inputs of the AND gates and another from the output of the AND gate of that stage and so on. This way, finally when it comes to the final stage then only output of one switch is high and this is used to identify the longest prefix amongst all the matched ones with the help of an encoder unit.

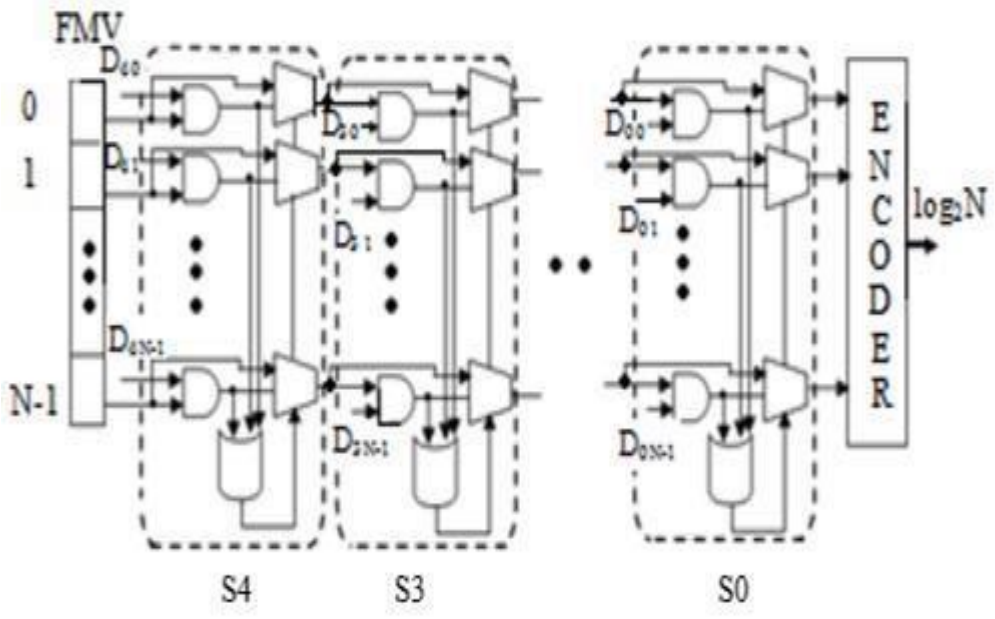


Fig. 4.1.3: Hierarchical structure for LPM

We are trying to enhance its performance by implementing pipeline to this structure. We have implemented the above structure through model sim and now we need to provide delay at every gate and implement the pipeline using trigger. That part still needs to be done.

4.2 Software based implementation

Now coming to the software based implementation we have mixed the TBM structure along with binary trie. We know majority prefix lie in the range of 16-32.So we search that part using binary trie and the early 14 bits are matched using TBM.

4.2.1 A Hybrid Structure

The T.B.M (Tree Bit Map) structure which is used in [5] is structure which takes the help of both I.B.M i.e. Internal Bit Map and E.B.M i.e. External Bit Map. This I.B.M helps to forward packet in case there is no matching in the super node. As after 24 bits the density of prefix is in the bits it simply uses very much space. Since storing of single prefix require a super node. Our Hybrid Structure is comprises of Nodes and binary trie. The Super Nodes uses E.B.M and not the I.B.M this structure is used till 14 bits and the rest of the bits will be handed by the binary trie. The reason for choosing 14 bits is that the density of the prefix present is high from 14 to 24 bits. The Prefix of length less than the 14 bits will be routed to the default router. So there is no need for the I.B.M in the hybrid structure. For Every incoming packet it is passed to the root super node. For traversing in the Super Nodes of hybrid structure it takes 3 bits at a time from most significant bit to least significant bit. Each time the 3 bits are matched with the E.B.M of that node. If the E.B.M value is **1** then it will proceed further. But if the value is **0** the packet will be router to the default router address. For traversing the binary trie it takes 1 bit at a time from most significant bit to least significant bit. For every **1** that it will encounter pointer proceed to right. For every **0** the pointer will proceed to the left. After a certain point when there is no link to proceed, the packet is routed to the last matched prefix.

4.2.1.1 Routing Operations

Insertion: The insertion process is applicable for the prefix which is greater than or equal to the length of 14 bits. The Insertion process is done as was mentioned above. The DIP is traversed from most significant bit to the least significant bit taking 3 bits at a time. Those 3 bits will be matched with the External Bit Map of the Nodes to find a way further. After the 14 bits it will be taking 1 bit at a time and moving to binary trie. There according to the bits scanned, as for **0** it will move left and for **1** it will move right. If there is no link to the next super node it will create a Super Node and change the External Bit Map of that Super Node. According to this way it traverses the hybrid structure. Once it reached the node for the prefix it will mark the node as **true** and store the router information in it.

Taking an example of the above process we will show how the certain prefix is inserted. However for understanding purpose we will use the Nodes till 6 bits and the rest will be handled by the binary trie.

Like for example if we want to insert **'10101101'** prefix with router information **K**. As for the Super Nodes it will take 3 bits at a time so initially it will be taking **101** then matching it with the

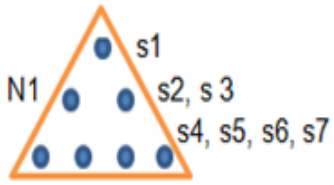


Fig. 4.2.1.1: Initial Structure

External Bit Map. The current structure is empty so the EBM of **N1** is **00000000**. So it will create a Super Node to that **101** i.e. 5th digit of the EBM as **1** to indicate that there is link now. Then it will traverses to that newly created Super Node. Now it will take the next 3 bits i.e. **011**

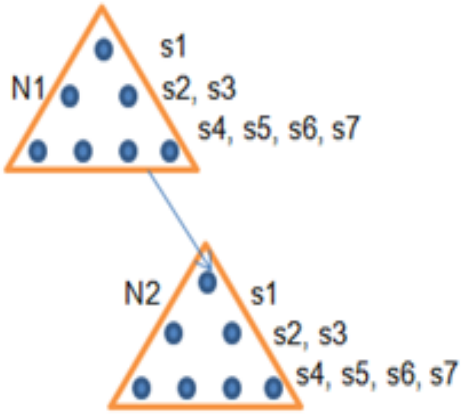
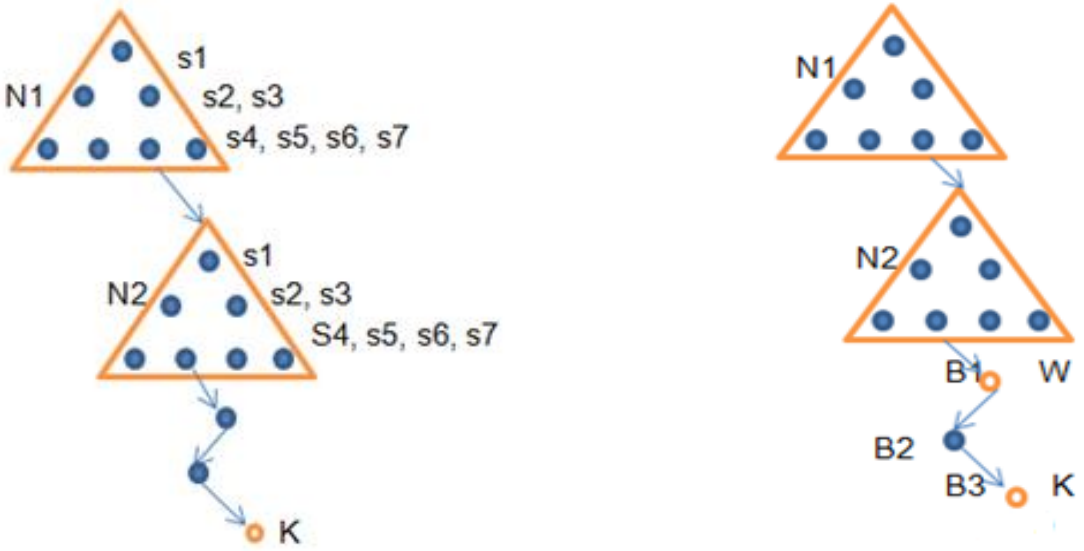


Fig. 4.2.1.2: Second Super Node Formed

check from the current super nodes EBM. Since it also does not have any link further thus it creates a binary trie node there and marking its EBM as **00010000**. After this point the pointer will take 1 bit at a time and then move accordingly as the bits encountered. Thus taking **0** it creates node to its left and the encountering **1** it will create a node in the right thus changing the structure as given. Then it will

mark the node as **true** and store the router address. For inserting another prefix **101011** with routing information **W** in the existing structure. The pointer does not have to create node as previous example. At first the first 3 bits are taken then matched to the EBM of **N1** as the 5th position is 1 (taking 0 as starting index) the pointer will simply move to the next Super node **N2**. Thus it takes the next 3 bits i.e. **011** it will then checks the EBM of this current node which also matches with EBM. So it will now simply mark the node



(a) (b)

Fig. 4.2.1.3: Insertion of new node

Searching: The data packet that comes for a match has a DIP (Destination IP Address) in its IP header. This IP header is of 32 bits in IPv4 and 128 bits in IPv6. As like the insertion

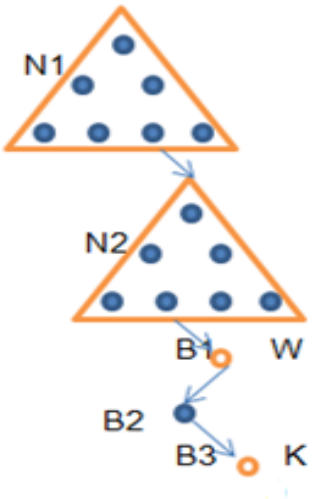


Fig. 4.2.1.4: Structure at a particular time instance

process for traversal in the super nodes 3 bits are taken at a time. Those 3 bits will be matched against the EBM of the super node. After 14 bits the DIP bits will be taken 1 at a time. Then the pointer moves as per the bit encounter. As for every **1** encounter the pointer will proceed to right. And for every **0** encounter the pointer will proceed to left. The pointer will store all the router data if there is any in the time of traversing the node. Then it will move forward as far it can proceed and when there is not any link to move it will return the last save router information. As for example suppose a data packet has arrived having a DIP of **10101101001010101010000101010010**. If we traverse the structure as per the example taken above the initial bits will be **N1** and matched those bits to its EBM. Then the next 3 bits will take it to binary trie. Since it encounter **W** router data and move from there towards left as **0** is encountered and the moves to right for **1**. There it found another router data so it discards **W** and saves **K**. As there is no possible way from there so it will now forward the data packet to the **K** router.

Deletion: The deletion process is like insertion process. The prefix is scanned from the left to right taking 3 bits at a time till we have super nodes. After that the binary trie starts and the prefix will be scanned 1 bit at a time. On reaching the node it just simply marks the node empty

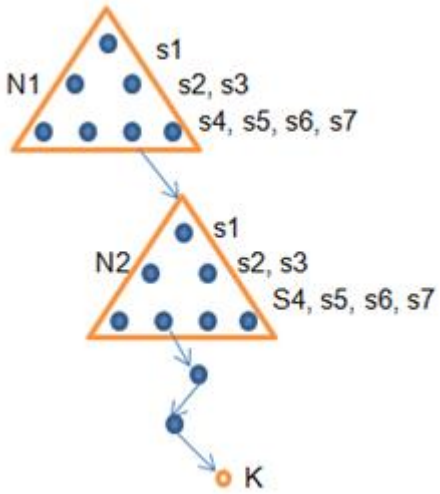


Fig. 4.2.1.5: Structure after deletion

or **false** and delete the existing router information stored in it. As for example if we want to delete the prefix **101011**. As per our example structure the first 3 bits will move the pointer to the super node **N2**. The next 3 bits make it point to the binary trie. Thus after reaching node **B1** it will delete the router information and mark the node empty or **false**. The following result is shown in the figure..

Chapter 5: Sample Code

5.1 Software based approach

Pseudo Code:-

//Root node N1 is defined with EBM value as **00000000 ,** Taking input from file to insert prefixes

Insertion:

//Initialising a pointer to root node

1. Taking 3 bits at a time. Repeat steps 1 to 5 till traversing within super-nodes.
2. Converting it to decimal.
3. Matching the EBM of currently pointing super-node.
4. If (EMB value == 1)
   Then point to the next super-node
5. Else
   a. Create a super node.
   b. Change EBM value at appropriate index to 1.
   c. Point to that node.
6. Taking 1 bit at a time. Repeat steps 6 to 8 till all bits of prefix are scanned.
7. Point to left node if encounter **0** and point to right node if encounter **1**
8. If (link not available)
   Create node then traverse.
9. Store the router address and mark the node for prefix storage.

Searching:

//Initialising a pointer to root node, Storing visited node information in a variable called **router**.

Taking 3 bits at a time. Repeat steps 1 to 5 till traversing within super-nodes.

1. Converting it to decimal.
2. Matching the EBM of currently pointing super-node.
3. If (EBM value == 0)
   Return default router address.
4. Else
   a. Point to next super-node.
   b. Save any router info if available.
5. Taking 1 bit at a time. Repeat steps 6 to 8.
6. Point to left node if encounter **0** and point to right node if encounter **1**
7. If (link not available)
   Return router.

Deletion:

//Initialising a pointer to root node

1. Taking 3 bits at a time. Repeat steps 1 to 5 till traversing within super-nodes.
2. Converting it to decimal.
3. Matching the EBM of currently pointing super-node.
4. If (EMB value == 1)
   Then point to the next super-node
5. Else
   Throw error of unidentified prefix.
6. Taking 1 bit at a time. Repeat steps 6 to 8 till all bits of prefix are scanned.
7. Point to left node if encounter **0** and point to right node if encounter **1.**
8. Delete the router information and mark the node as empty

Here is the main function of the program:

```
#include"projectMix.h"
#include<time.h>
#include<fstream>
int main() {
        initialise();
        string str = "";
        char ch;
        int cou;
        ofstream myfile;
        ifstream file;
   file.open ("E:\\LPM\\trie.txt");
   if (!file.is_open()) return 100;
   string word;
   while (file >> word) {
      file >> ch;
      ADDPREFIX(word , ch);
   }
        file.close();
        myfile.open("E:\\LPM\\trie.txt", std::ios::app);
        if (!myfile.is_open()) return 100;
```

```
        while(1) {
                int choice;
                cout<<endl<<"Press 1 to search"<<endl<<"Press 2 to ADD"<<endl<<"Press 3 to
Update"<<endl<<"Press 4 to delete"<<endl<<"Press 5 to Exit"<<endl<<"Enter your Choice :";
                cin>>choice;
                if(choice == 1) {
                        cout<<"enter the dest address"<<endl;
                        cin>>str;
                        cout<<"The Address is "<<Search(str);
                }
                else if(choice == 2) {
                        cout<< endl << "enter the prefix then the router name :";
                        cin>>str;
                        cin>>ch;
                        ADDPREFIX(str , ch);
                        myfile<<"\n";
                        myfile<<str<<"     "<<ch;
                }
                else if (choice == 3) {
                        cout<< endl << "enter the prefix then the router name :";
                        cin>>str;
                        cin>>ch;
                        ADDPREFIX(str , ch);
                }
                else if(choice == 4) {
                        cout<< endl << "enter the prefix to delete :";
                        cin>>str;
                        Delete(str);
                }
                else
                        break;
        }
        myfile.close();
   return 0;
}
```

5.2 Hardware based approach

Program for MUX:

```
module mux(select,d,q);
  wire q;
  wire select;
  wire [0:1] d;
  assign q=d[select];
endmodule
```

Program for OR gate:

```
module OR4(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,o);
    assign o=a0|a1|a2|a3|a4|a5|a6|a7|a8|a9;
endmodule
```

Program for implementing the modules:

```
module S3(lt,lt1,lt2,lt3,lt4,fmv,or2,o);
  input [0:9] lt3;
  input [0:9] lt4;
  output or2;
  output [0:9]o;
  for(i=0;i<=9;i=i+1)
  begin
    assign a[i]=fmv[i]&lt[i];
  end
  OR4 b(a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],or2);
  assign or3=or2;
```

```
assign d[0]=fmv[0];

assign d[1]=a[0];

mux c(or3,d,o4);

assign o[0]=o4;

assign d1[0]=fmv[1];

assign d1[1]=a[1];

mux c9(or3,d1,o5);

assign o[1]=o5;

assign d2[0]=fmv[2];

assign d2[1]=a[2];

mux c1(or3,d2,o6);

assign o[2]=o6;

assign d3[0]=fmv[3];

assign d3[1]=a[3];

mux c2(or3,d3,o7);

assign o[3]=o7;

assign d4[0]=fmv[4];

assign d4[1]=a[4];

mux c3(or3,d4,o8);

assign o[4]=o8;

assign d5[0]=fmv[5];

assign d5[1]=a[5];

mux c4(or3,d5,o9);

assign o[5]=o9;

assign d6[0]=fmv[6];

assign d6[1]=a[6];
```

```
  mux c5(or3,d6,o10);
  assign o[6]=o10;
  assign d7[0]=fmv[7];
  assign d7[1]=a[7];
  mux c6(or3,d7,o11);
  assign o[7]=o11;
  assign d8[0]=fmv[8];
  assign d8[1]=a[8];
  mux c7(or3,d8,o12);
  assign o[8]=o12;
  assign d9[0]=fmv[9];
  assign d9[1]=a[9];
  mux c8(or3,d9,o13);
  assign o[9]=o13;
  for(i2=0;i2<=9;i2=i2+1)
  begin
  assign o14[i2]=o[i2];
  end
  for(i3=0;i3<=9;i3=i3+1)
  begin
    assign a1[i3]=o14[i3]&lt1[i3];
  end
endmodule
```

Now we need to call this module 5 times in order to implement our structure.

Chapter 6: Testing, Results, Discussion on results

Processing time: The processing time can be calculated by number of clock cycle taken times time of each clock. The Best Case is when the prefix is of exact 14 bits. Then the pointer only has to traverse the super nodes and it takes only 4 clock cycles to traverse to the leaf of the last super node. The worst case is that when the only prefix is of 32 bit length then pointer has to traverse whole super node and the whole binary structure of 18 bits after that. This whole traversal takes 22 clock cycles. In the average case it takes 11 clock cycle. Thus according to this data the following are the time analysis.

- Best Case : 1.08 ns
- Average Case : 3.51 ns
- Worst Case : 5.9 ns

The throughput of the following structure is $16.9 * 10^7$ packets per sec.

System Specification: Intel i3, 32 bit, 4GB RAM, 3.70 GHz

We have implemented the hardware based structure usind Verilog in modelsim. We are giving the length part and the fmv i.e the final match vector as input. Now if the and result of any of the bit of length bit and the fmv is one,the new anded output will serve as fmv of next module or else same previous fmv will be sent forward and will be anded with next length bit. Here is the waveform of output which we are getting:

The input length vectors are :

Lt: 1100101110

Lt1:1111000011

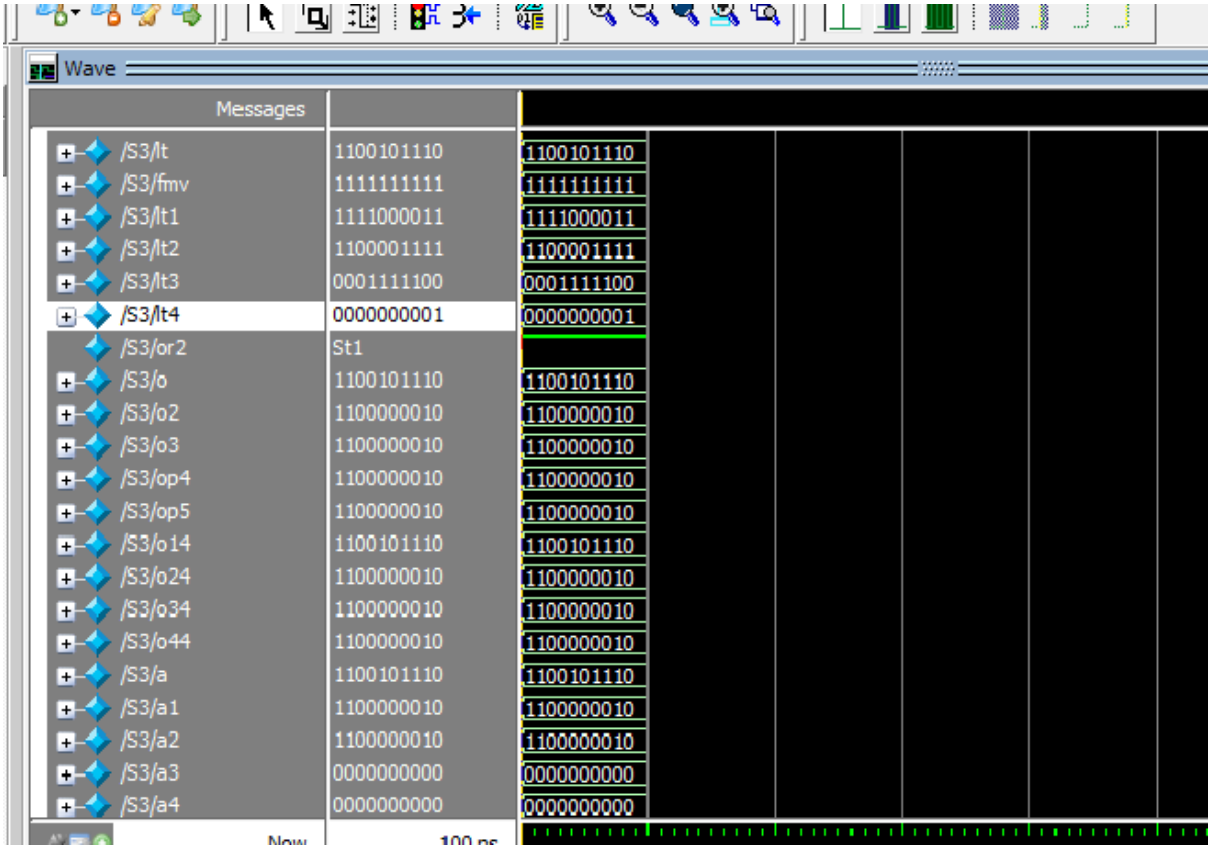
Lt2:1100001111

Lt3:0001111100

Lt4:0000000001

The fmv we passed is: 1111111111.These are the 10 prefixes we passed. Longest prefix is stored in o5.

Waveform from hardware implementation of the modules:



Chapter 7: Conclusion

The existing hardware based architecture for longest prefix match is fast but needs to be in sorted order. Some of those hardware architecture which do not need the prefixes to be sorted suffer power and cost issues. There are some software based solutions like trie data structure but minute changes and updating causes problem as we need to re build the trie. The depth of the trie is also an issue, more the depth more the search time and slower the architecture gets .So this introduction of hybrid structure helps and solve our problem of speed, cost and efficiency. Along with that pipelined based techniques provides higher throughput along with low register and low power consumption.

Chapter 8: References

[1] "Survey and Proposal on Binary Search Algorithms for Longest Prefix Match" - Hyesook Lim and Nara Lee , Ewha women's university

[2] "Survey and Taxonomy of IP Address Lookup Algorithms" - Miguel Á. Ruiz-Sánchez, INRIA Sophia Antipolis, Universidad Autónoma Metropolitana Ernst W. Biersack, Institut Eurécom Sophia Antipolis Walid Dabbous, INRIA Sophia Antipolis

[3] Engineering Networking – Stanford University – (https://lagunita.stanford.edu/courses/Engineering/Networking-SP/SelfPaced/info)

[4] "Longest Prefix Matching using Bloom filters" - Sarang Dharmapurikar Praveen Krishnamurthy David E. Taylor - Washington University in Saint Louis 1 Brookings Drive Saint Louis, USA

[5] "On Fast Address- Lookup Algorithms" – H. H. Tzeng and T. Przygiend,University of Florida

[6] IP Address Lookup Algorithms Harrick Vin, Department Of Computer Science, The University of Texas at Austin.

[7] V. Srinivasan and Varghese "Faster IP Lookups using Controlled Prefix Expansion," Proc. ACM SIGMETRICS '98, June 1998.

[8] M.A. Ruiz-Sanchez, E.W. Biersack, W. Dabbous, Survey and taxonomy of IP address lookup algorithms, IEEE Network 15 (2) (2001) 8–23

[9] P. Newman, T. Lyon, G. Minshall. "Flow Labelled IP: A Connection less Approach to ATM." Proc. IEEE INFOCOM' 96, pp. 1251-1260, San Francisco, USA

[10] Y. Rekhter, B. Davie, D. Katz, E. Rosen, G. Swallow. "Cisco Systems' Tag Switching Architecture Overview." RFC 2105, February 1997

[11] Lih-Chyau Wuu , Tzong-Jye Liu, Kuo-Ming Chen, "A longest prefix first search tree for IP lookup," Computer Networks 51 pp. 3354–3367, 2007.

[12] L.C. Wuu, S.Y. Pin, A fast IP lookup scheme for high speed routers, Journal of the Chinese Institute of Engineers 28 (3) (2005) 521–533

[13] M. Kobayashi, T. Murase, A. Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing", IEEE International Conference on Communications, pp. 1360 - 1364 vol.3, 2000