Broadband Networks
Prof. Karandikar
Department of Electrical Engineering
Indian Institute of Technology Bombay

Lecture No.22
IP Addressing Lookup and Packet Classification

So till now we have studied the IP addressing structure and we have seen that IP addresses addressing scheme can be of two types.1 is the classfull IP addresses and the another one is the classless IP addresses which uses the classless inter domain routing or citr.

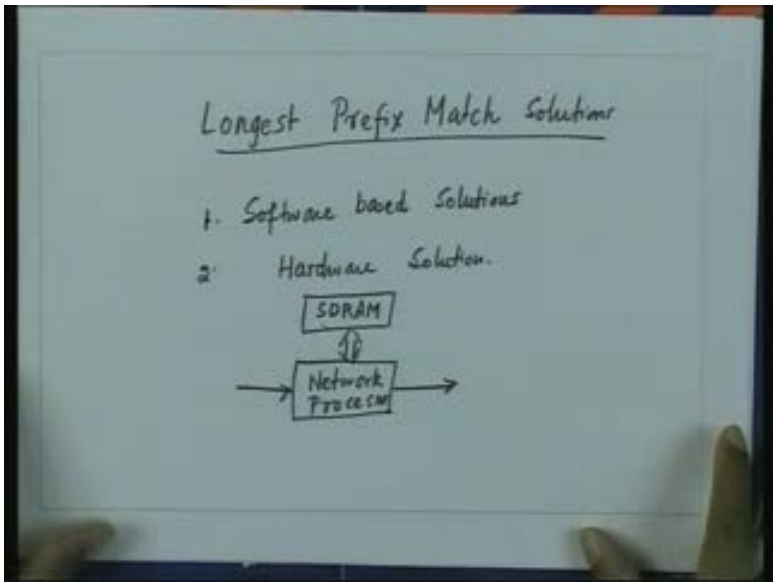
As we have seen that by using classless inter domain routing, not only we can address the problem of IP address space efficiency but we also retain the property of scalability and making the size of the forwarding table to be compact, but as we have seen now that when an IP packet comes and we have to look up the destination IP address in the forwarding table then if we have used classfull IP addressing scheme, then the search was very simple you simply need to perform you simply first need to determine whether a destination IP address is a class A IP address or a class B IP address or a class C IP address and then need to perform an exact match with the network part of the destination IP address with the corresponding entry in the forwarding table.

However, when we use a classless inter domain routing then we have seen that due to the aggregation at arbitrary lengths the network numbers are no longer fixed but they are actually the address prefixes which can be of arbitrary length. Therefore, it is possible that given destination IP address may match several of the entries in the forwarding table and the forwarding needs to be done corresponding to that entry which gives you the longest match and this is what is called as the longest prefix match. Now this is one of the reasons for performance bottlenecks in internet router and today we will actually look at some of the mechanisms that have been used for the longest prefix match.

Now, in short typically the IP address look up in an internet router is either done by using a combination of network processors or a memory. In the memory, you typically store the routing or forwarding database and a the search algorithm runs on the network processor and the performance metric for an IP address lookup algorithm as we already seen is there are two performance metric1 is the search time. So, search time indicate how much time or how much memory accesses you need to make in order to perform a lookup on a destination IP address. So that is search time and second one is the update time. By update time, we mean that whenever a change in the forwarding database occurs how much time it takes to update the forwarding table entry.

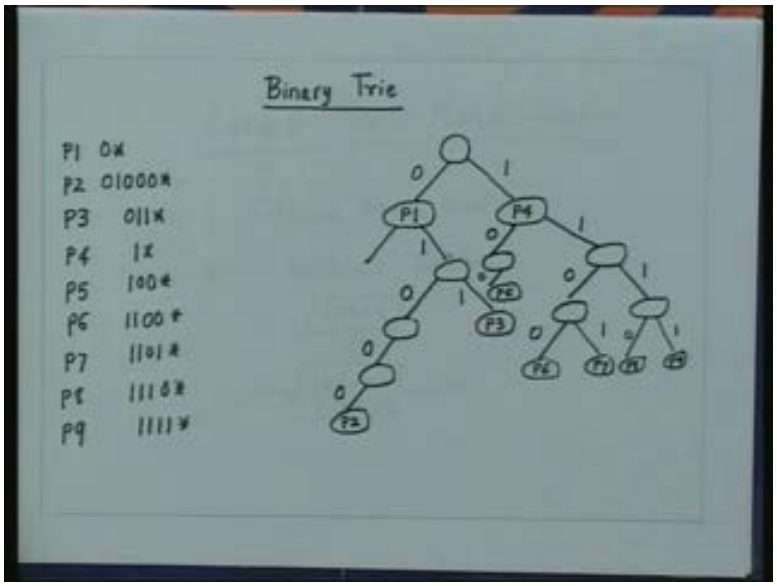
So these two metrics govern the performance of a given IP address lookup algorithms. So today what we will do is that we will look at some of the simple address lookup algorithms that have been proposed in the literature for the lookup of a destination IP address and that are currently in use in the internet routers.

(Refer Slide Time: 04:47)



So we are looking at the architecture is something like this that the network processor will run a search algorithm and the forwarding database is stored in an sd ram. So, now let us look at some of these look up algorithms and simplest one we will look up as the trie algorithms. So let me just explain binary trie algorithm.

(Refer Slide Time: 05:12)

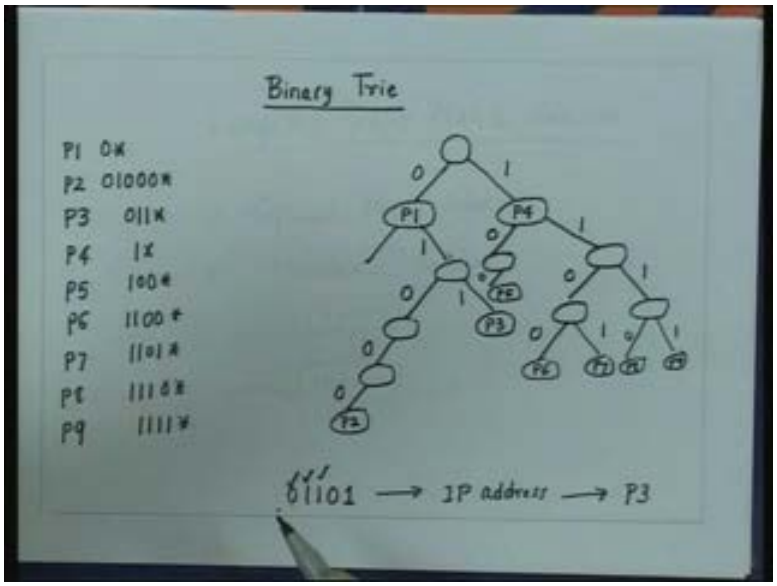


Now in binary trie the routing database is stored in the form of a track. Now to explain this, let us consider an example prefixes. So we will consider that p 1 is you know 0 star p 2 is 0100 star then p3 is 0 1 1 star. These are some of the example prefixes p4 is 1 star p5 is 100 star p 6 is 1100 star p 7 is 1101 star and p 8 is 1110 star and p 9 is 1111 star.

So there are about nine prefixes table and you know you can assume that a destination IP address is of 8 bit or 6 bits. You know the longest prefix length here is about five and the shortest prefix length is 1. So when you organize this tree structure, so we see it is a binary tree. So we start with 0 1 here. Now here, we get p1 which is you know 0 star is stored on this intermediate nodes and here we will get p4 which is you know 1 star, then we further do branching here the left and the right and the left. So 0 0 star. There are no such entries. So there the trie the tree does not exists here.

On this side, we have one here we get an intermediate nodes here and one further which will give me an entry corresponding to p3 which is 0 1 1 star so 0 1 1 star is stored here. If we had on the left branch a 0 then another intermediate node a 0 and another intermediate nodes with 0. So finally we get 0 1 triple 0 star which is actually giving p2, then p4 is then further branches p4 which is from 0. This is from 0 left entry and another left entry which is further 0 star giving you p5 and here on one we have an intermediate node again 01 here. So here this is 1 1 0 0 star which is p6 and 1 1 0 1 star which is p7 and then we have here p8 and p9. p8 is 1110 star and p9 is 1111 star p9. So now this is how you know what happens is that, for the given example. The database the forwarding database is organized in the memory in the binary trie form which is actually a tree structure, now suppose the destination IP address something like 01101.

(Refer Slide Time: 09:26)

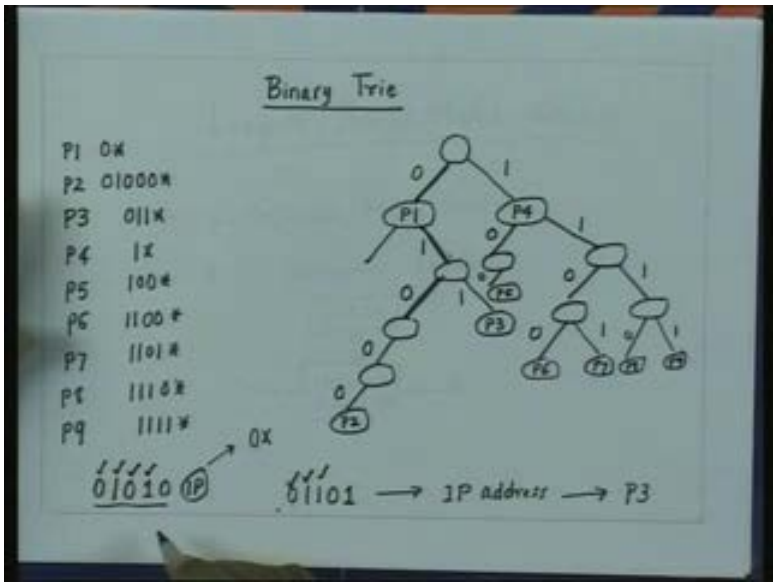


So this is let us say five bit given destination IP address. For the purpose of example we can assume that IP address is a five bit IP address. Now the first bit is examined which is 0 all right. So you take left branch here you know and you reach this intermediate nodes. Now when you come here you remember that this node is storing a prefix p1. So that you know if later on if you do not find a search, you will preserve the search to be the p1. So now you remember that you have encountered a prefix which is p1. Now the next bit happens to be 1 here. So you will make a right here. So you come to the intermediate node. This of course is not a prefix. Now you will examine, so this bit you have examined first bit, second bit. Now you are examining the third bit and you encounter here p3 and which happens to be a leaf node.

So this destination IP address will then declare that it matches the entry which is you know p3 and so then you will fetch on corresponding to this p3 entry it will point to which forwarding interface that packet needs to be forwarded and the router will accordingly then forward the packet. Now you can see here that if this routing database is stored in the form of a binary trie structures and for the example destination IP address you need to made you need to make three memory accesses so first time you go to 0 here and second time you go for 1 here and the third time you for 1 here. So you know if w is the length of the IP address, then in the worst case you will have to make order of w accesses to the memory.

So as you know that in the case of an internet router, the IP address is going to be 32 bits. In this example, we have taken the IP address to be five bits but in an internet router the IP address will be 32 bits. So, in the worst case you will have to make 32 memory accesses and now this can prove to be very costly because typically this routing database will be stored in an external memory in a d ram and even it is a synchronous d ram sd ram or a dd ram which are faster you will still in the worst case will have to make 32 memory accesses leading to a large search time that is one issue that we need to address. Now let us take another example:

(Refer Slide Time: 12:15)



Suppose if you had an IP address which is 0101, this is 4 and 5. So let us say this is the IP address. Now here the first bit is 0. So you will make a left you know a left branch here. So you have examined this. Now you remember that p1 is the longest prefix match p1 is the prefix match that has been found so far, second bit is one, so you will make right turn here. Now this is this is an intermediate node which does not store any prefix. So p1 still continuous to be your match, the third bit is 0. So you will make now a left.

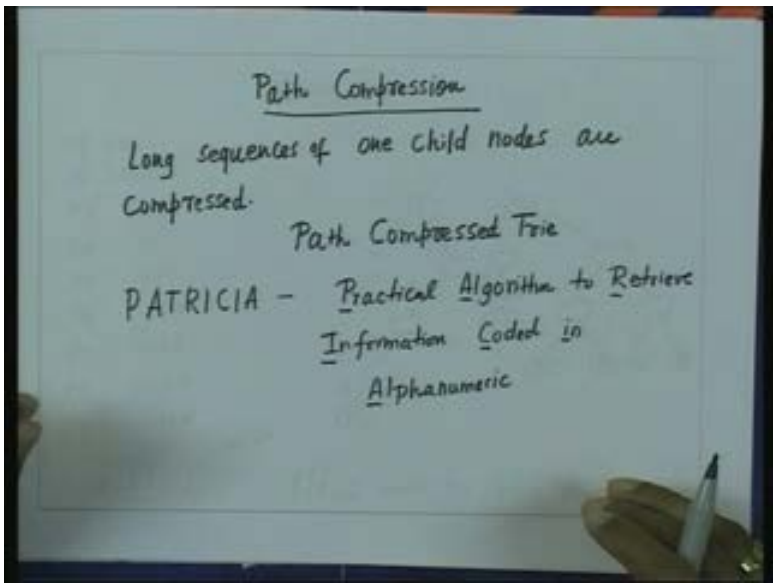
This is again an intermediate node here. So your match found so far is p1. Now, the fourth bit when you go for examine that is one and there is no there is no right entry for corresponding to1. So therefore you know you declare that the match found so far is p1 and therefore this IP address matches the entry of you know 0 star that is p1 and then you retrieve the forwarding interface

corresponding to the entry p1. So, this is how you know you can you can do a search but as i just discussed the problem that is w is the length of the IP address then the worst case search time or the worst case memory accesses that you may need to may is order w. So we will just you know see how we can how we can address this problem but1 more problem that you need to see here is that that in this given binary tree.

There are this long sequences you know there are this long sequences of one child nodes you know for example: this intermediate nodes this intermediate nodes has just one child node which is this. This again this intermediate node has this one child node and these1 child node consumes large additional memory. So therefore, we must find some mechanisms to compress this trie, so that you know not only we can have an efficiency in terms of saving the memory but perhaps can also you know improve the search time. So we will you know look at some of the methods which have been used to compress this one child nodes trie and one of the technique which is used is called you know path compressed trie. So let us look at the example of what is called as path compressed trie.

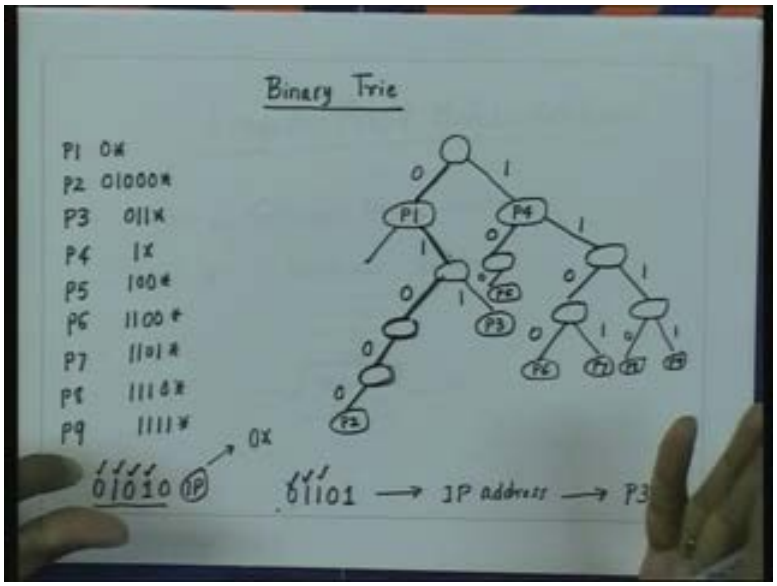
So in path compression what is done is that long sequences so wherever you know if in tries if long sequences of long sequences of one child nodes are compressed. So look at you know the path compressed trie. This algorithm is also called a Patricia algorithm which is actually a short form for practical algorithm. So, practical algorithm to retrieve information coded in alpha numeric.

(Refer Slide Time: 16:43)



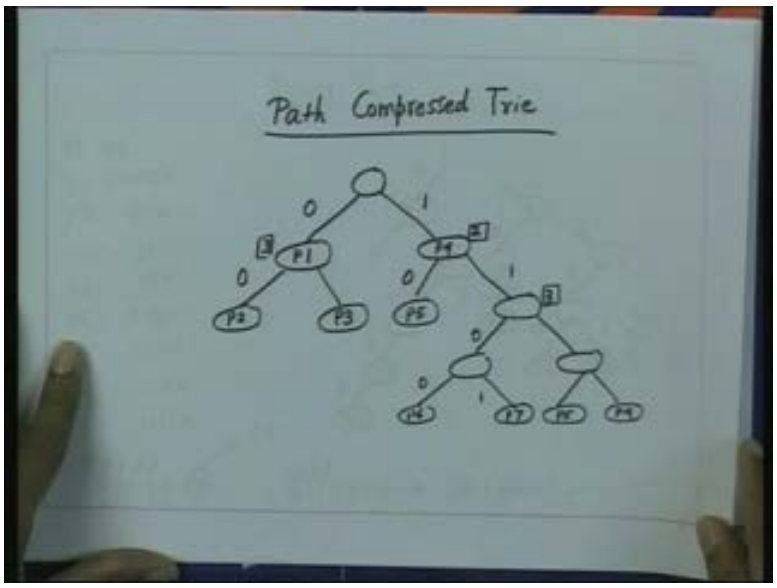
So, this is how you know a Patricia algorithm. Now this algorithm has been used in the UNIX based d for the IP addresses lookup algorithms and we will take the same example that we have just taken for explaining the binary trie and see how a path compression works. So for the same example you know which we had taken here this example.

(Refer Slide Time: 17:06)



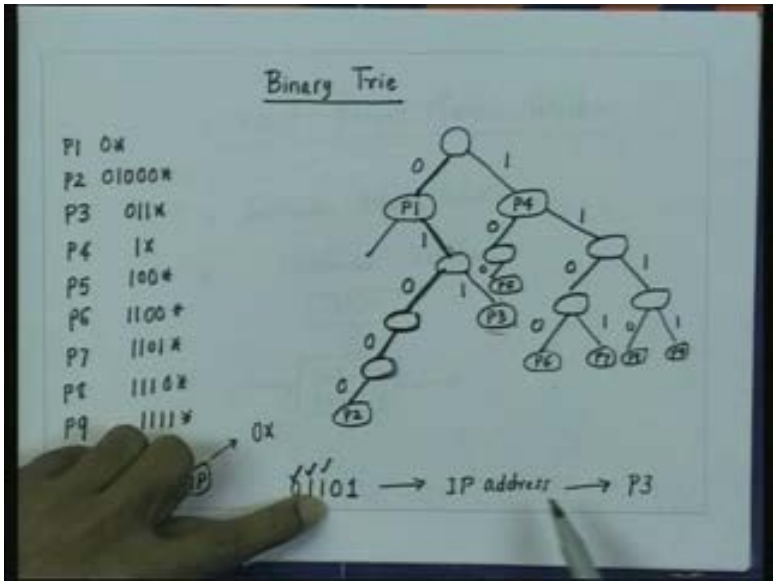
Now we see that these sequences of ones that can be compressed here and similarly we can compress you know these sequences. So, as a result we have a path compressed trie which looks which will look like the following. So we have 1 here and we have compressed this p3 p4 here p5 and this remains same as there is no compression here. So if you look at you know these two we have seen that these path we have compressed. Now you see here that since we have now compressed the path, we also indicate that this intermediate node which bit now needs to be examined that how many bits you know they need to be skipped.

(Refer Slide Time: 18:55)



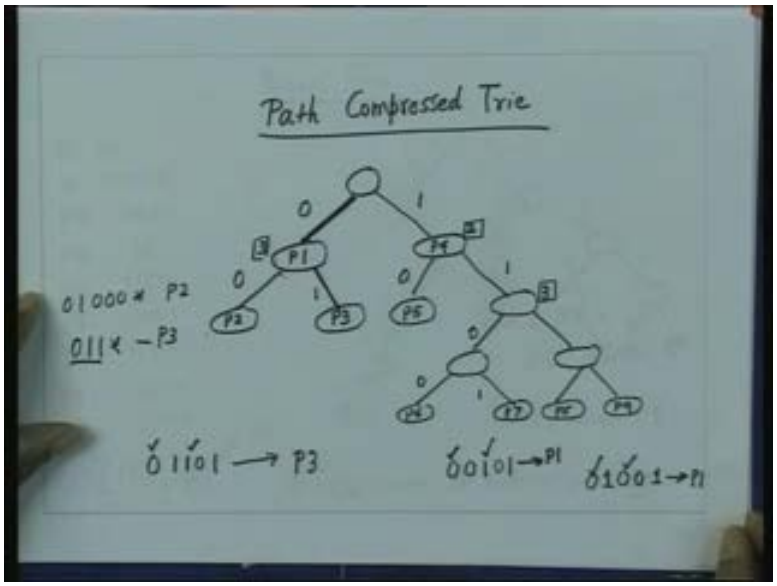
So what we do now is that, so this indicates that you know that this indicates here that which bit needs to be examined here. So what you do is, let us do let us do is a search which was the example which we had taken as 01101 right.

(Refer Slide Time: 19:24)



So in this particular case you know in this example what we have done is that, we took we took a left turn which is a 0 in here p1 we remember that p1 is a match then we took a right turn here so which was1 now this1 is an intermediate node of course. So our match found so far just p1. We took as we examining now third bit and we found it1 and when we found this one then p3. Now is now longest match and not p1 and therefore you know and p1 and p3 happens to be the leaf node. So we will declare that this particular IP address matches p3.

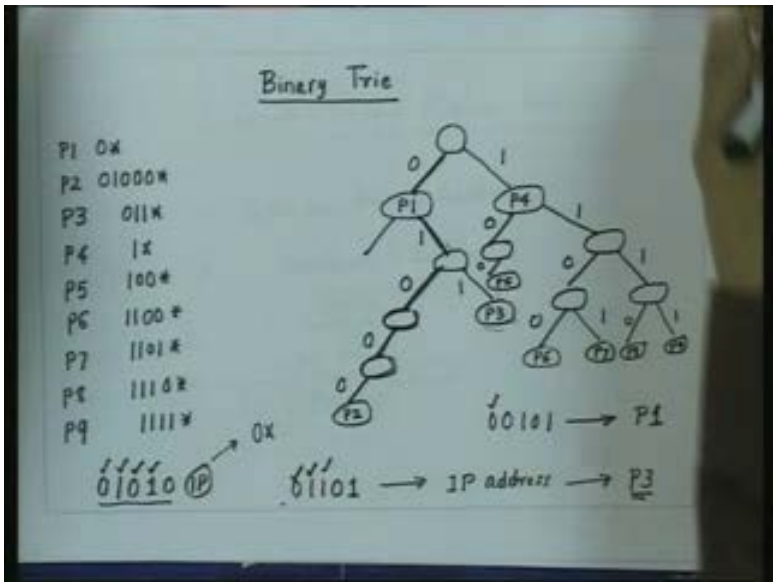
(Refer Slide Time: 20:04)



Now you look at this the same the same IP address we have to examine. Now in this case take the first bit which is 0 and we come to p1 p1 is of course match found so far but now we will examine you know the third bit. So, the third bit happens to be one, if the third bit was 0 then we could have taken the left turn but the third bit happens to be you know 1 so we will take a right turn and we will come to p3. Now when we come to p3 in the Patricia algorithm, the path compressed tree it is important that when we reach this leaf node we perform an exact match with the prefix which is been stored at p3 because we had skipped the previous bits we need to perform an exact match at p3 to confirm whether this destination IP address indeed matches the prefix at p3 or not.

So, now the prefix at p3 happens to be you know 011 star and we match here and we find that yes there is a match with the exact match with this IP address prefix and declare that this you know matches the prefix p3. Now why this is important? This point is important that we have to perform an exact match exact match, then now let us look at the address which have been like 0 01 01 so if this has been the address then the left bit matches with the with the 0 then look at let us let us look at this IP address how the search would have been performed in ordinary binary trie.

(Refer Slide Time: 21:54)



So let us look at the addresses as 00101. So, how the search would have been performed here. We took a left turn, if we took the left turn then we reach a node where p1 prefix is stored so the match found so far is p1. Now the next bit is 0 and since there is no left turn here so this destination IP address clearly matches the entry p1. Now let us look at how the search would be performed in path compressed trie. We take a left turn here, we come to p1. p1 is the match found so far. Now will examine the third bit third bit is1 so therefore we take a branch to p3.

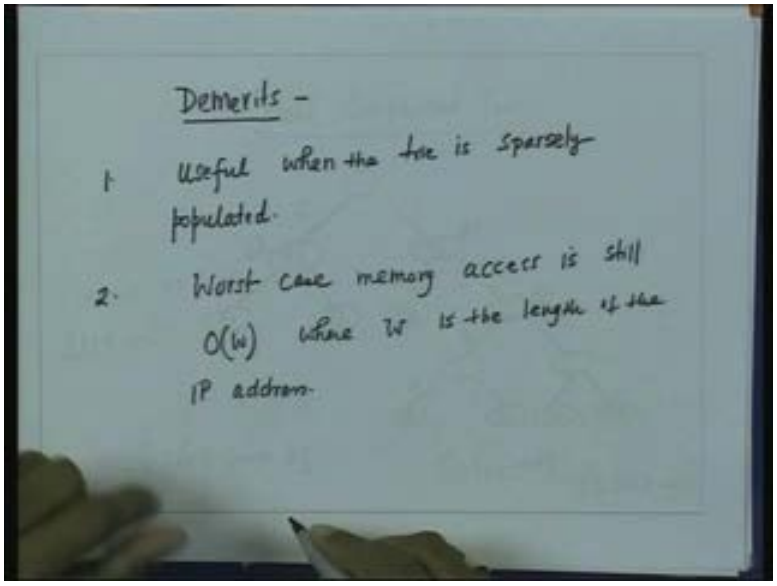
Now this is the leaf node. If we find, now we will do an exact match with 011 which fails so therefore we will backtrack and remember that p1 is we will remember that p1 is the is the is the match. So let us take some of another example. So another example corresponding to p2 would be let us take an example which is 01001. So in this case, this matches the IP address in p1. Now in this case, will do a first match with 0, so we have found a match here. We remember the match has been found to the prefix p1. Now we will match the third bit third bit here is 0. So we come to p 2.

Now when we come to p 2, we perform an exact match with p2 and with the p2, the p2 address has this exact match to be, so the p2 happens to be so this is your p3 and your p2 happens to be 01 0 0 0 star. So, we find that this does not match with the p2. So, even for this you know, even for this, the match that has been found as p1. So as you can see that in some cases, an additional search may have to be done in the path compressed tries but in general path compressed trie will actually reduce the size of the memory that is required to be stored by eliminating the long sequence, long sequences of child nodes and as you can see here, that even in the path compressed trie.

The worst case memory exercise that will be required to done, it is still the order of the size of the IP address. So, if you see the disadvantages the demerit of the path compressed tries are that are you know this is really useful the path compressed trie is really useful when tree is sparsely populated and then actually it will be more effective and an unfortunate and of course the worst

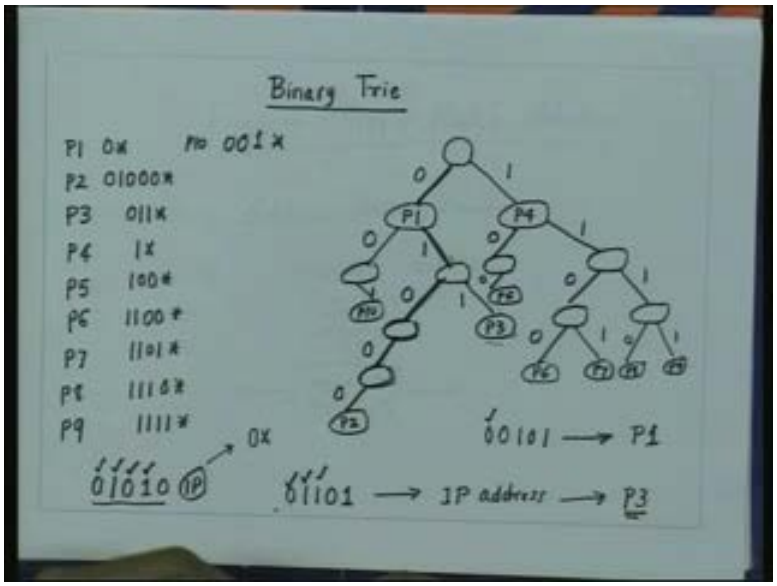
case memory accesses are still worst case memory accesses is still order w where w is the length of the IP address.

(Refer Slide Time: 26:27)



Now so after the path compressed trie or the Patricia algorithm, several algorithms have been proposed really to organize the database in such a manner that the search time is reduced to the updates time is improved. Now note that update in the binary trie is very simple for example.

(Refer Slide Time: 26:54)

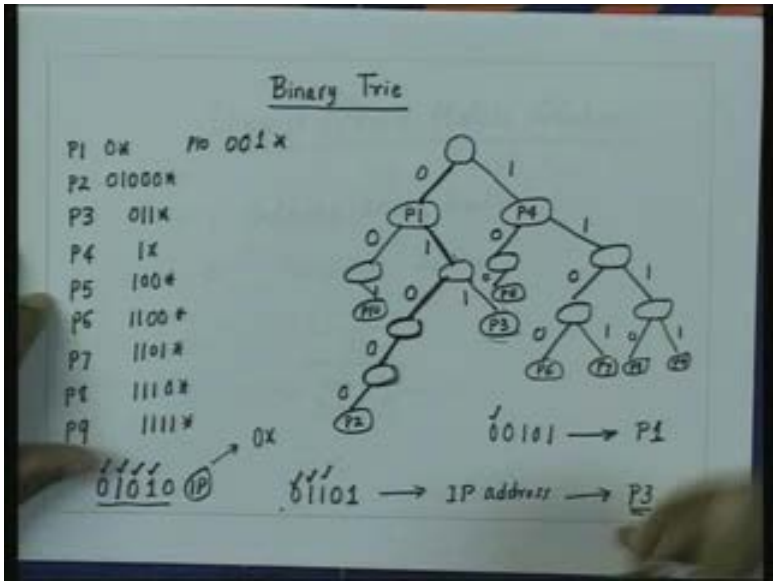


Let us say that because of the changes in the routing database, let us say that a new prefix needs to be added and that new prefix is something like say p10 and p10 happens to be 001 star. So it is

very easy then you add a node here which is 00 and then add 1 star here. So that becomes you know p 10. So you will add two nodes here by putting a pointer here and another pointer here so you know you can add a node 10 here. Similarly, deleting a node is very easy. Suppose, this p2 gets deleted and p2 gets deleted and then you know simply you can remove these intermediate nodes.

Similarly here the p 6 gets deleted. You can remove all these intermediate nodes. However, this p1 gets deleted then you can simply convert this into an intermediate node without a prefix but retain it because you know the p3 and p2 are still there. So, therefore addition and deletion of nodes in the binary trie is very simple. However, you know the disadvantage continues to be same that the number of memory accesses that are required is of the order of the length of the IP address. So, we will look for some more algorithms which have been proposed to improve to improve the search time. So let us look at another trie which have been proposed is called multibit or expanded trie. Now the basic idea of the multibit trie was that if so in a prefix table, there will be several prefix length you know in a typical router the prefix length can expend from 1 to prefix length of 32. As we have just seen in this example that, there were in this example there where prefixes of length 1 prefixes of length 3 prefixes of length 4, prefixes of length 5.

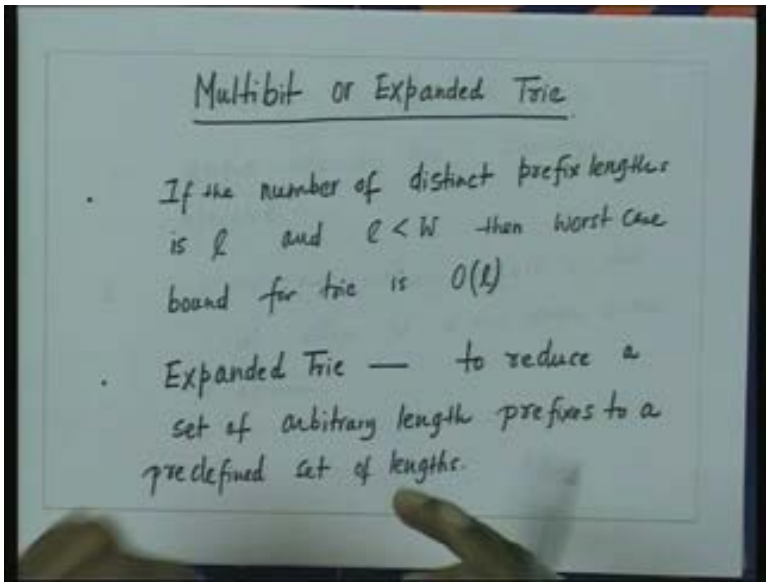
(Refer Slide Time: 29:28)



So all the prefixes were you know existing. So the basic idea is that if the number of distinct prefix length, if the number of distinct prefix length lengths is 'l' where 'l' is much less than 'w' then it can be shown that the worst case bound, in the worst case bound for the trie will be order 'l' and not really the order w.

So what is the basic idea is that we form a trie which is called expanded trie. So in expanded trie, the basic idea is to reduce a set of or the database of arbitrary length prefixes arbitrary length prefixes to a predefined set of length and then so what we do really is that, in a routing database we can have prefixes of length 1, prefixes of length 2, prefixes of length 3, prefixes of length 4 and so on up to prefixes of length 32 you know.

(Refer Slide Time: 30:52)



Some prefix length may not be there. For example: prefixes of length 8 or prefixes of you know 16 or 10 or 12. These may not be there. Now the basic idea of expanded trie is that we fix certain distinct prefix length. For example: in the example that we are just considering, we just considered for the binary trie where the prefixes of length 1 2 3 4 5 were there. We may say that, we will expand some of the ,so we will keep the distant prefix length to be let us say only 3 let us say prefixes of length 4, prefixes of length 2 and prefixes of length 5.

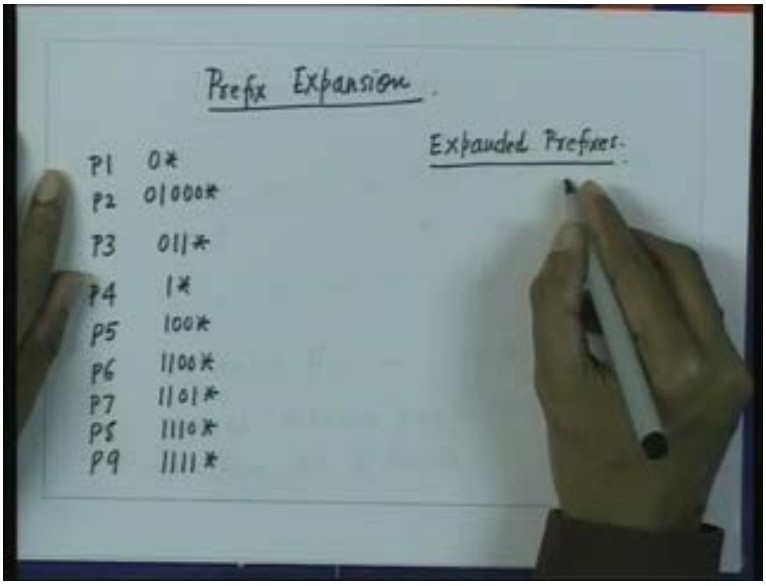
Now if there are prefixes of length 3, then we will expand them to prefixes of length 4. If there are prefixes of length 1, we will expand them into prefixes of length 2. So what we will do is that will have only three prefix length. Prefixes of length 2, prefixes of length 4 and prefixes of length 6. Now the question really is that, how do we define these distinct length whether it is to be two or it should be three or you know that question we will answer later, but suppose you know, we work out that we will expand these set of prefixes into set of distinct prefix length which we have considered that which we will consider this example to be prefixes of length 2, 4 and 6 and then organize a multibit trie instead of a binary trie we will organize a multibit trie.

In binary trie, we were always you know taking a branch decisions based upon bit by bit inspection that is what we are taking in binary trie. In a multibit trie, we will take a branching decision by looking at two or three or four bits simultaneously. So by looking based upon multibit inspection, instead of binary inspection. Based on multibit inspection that is, we will take a decision branching decisions that looking at two or three or four bits simultaneously. So by looking based upon multibit inspection instead of binary based on multibit inspection, we will take a decision to branch. So, let us consider an example and then clear up how the expanded trie or a multibit trie would look like.

So let us look at over our original prefixes you know i will explain first the idea of prefix expansion. So, let us say that we had this original prefixes which were p1 0100 star p3 was 011 star p4 was1 star p 5 was 1 0 0 star p 6 was11 0 0 star p 7 was11 01 star p8 was 111 0 star and p9

was 1111 star. Now let us look at you know how do we expand. So, these are our expanded prefixes.

(Refer Slide Time: 34:32)



So let us say we want to expand into prefixes of length 2, prefixes of length 4 and prefixes of length 5. So how do we get prefixes of length 2? So the two prefixes are there which are of length1, we will expand them into prefixes of length 2. So we will have 00 star. This will corresponds to p1 and we will have 01 star which also corresponds to p1, then we will have 1 0 star which will again p1 oh sorry p4 and then 11 star which will again be p4.

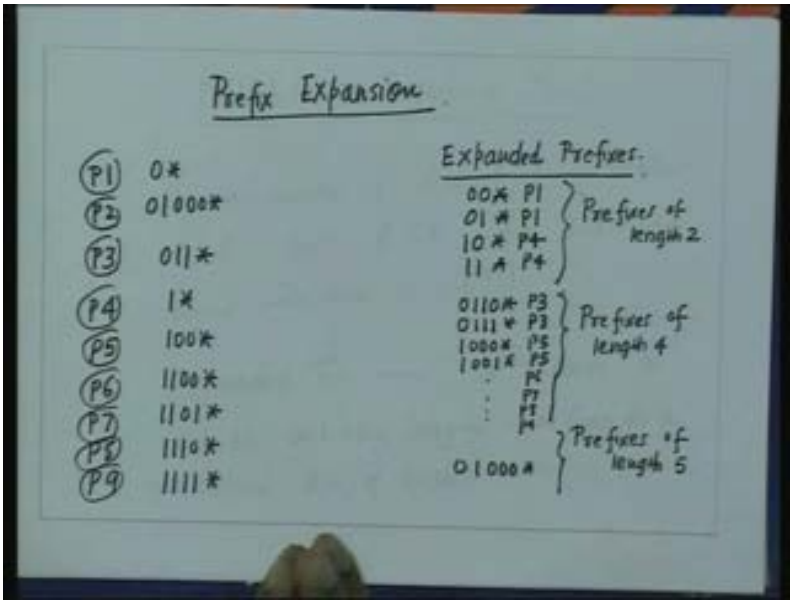
So, these two are prefixes of length 1. They have been expanded into four prefixes of length two. Two of them correspond to p1 and two of them correspond to p4 and then let us say prefixes of length four. So prefixes of length four you know already we have p6 p7 p8 p9. Now p6 p7 p8 and p9, these are already prefixes of length four. Now we need to consider this p3. We need to p3 expand into prefixes of length four. So let us say 011 so i will have 011 0 star which will correspond to p3 and then 0111 star which will again correspond to p3. Now i will take p5. p5 is1000 star which will again be p 5 and 1001 star which will again be p5 and of course you have p6 p7 p8 and p9. These are already prefixes of length 4.

Prefixes of length 5, there is only 1 prefix which is p2. So, i will have 01000 star. So what we have done is that, we have expanded the basic trie. We have expanded this set of database into three types of distinct prefix length. One is of prefix length 2, another set has prefixes of length 4 and third have prefixes of length 5. So, now let us see how we can make a trie or a multi multibit trie corresponding to this. Now first level is you have prefixes of length two, then we will have prefixes of length four and third one will have prefixes of length five. So, now let us see how it looks like.

The expanded trie: for the same example so, now what we will do since we have since there is no prefix of length one, there are only prefixes of length 2 only. So we will do the search for the

first two bits instead of doing the binary bit by bit. Now we will do the search on a group of bits level on a multi bit level.

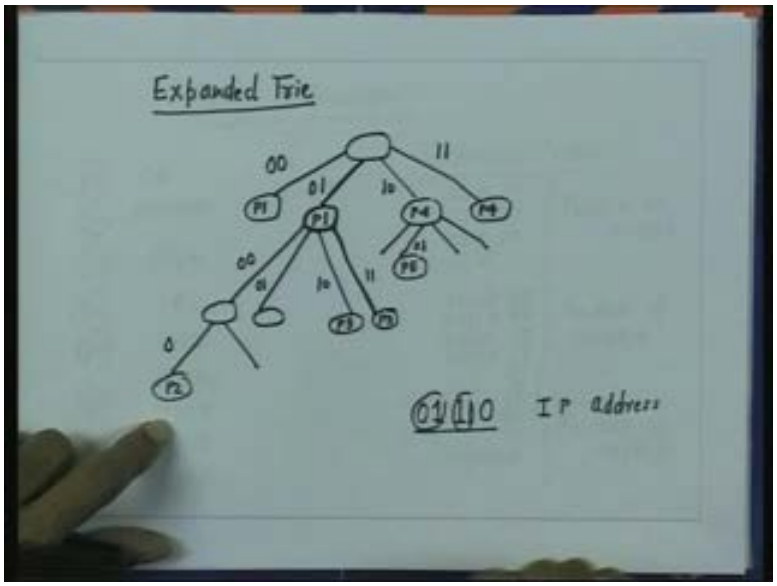
(Refer Slide Time: 38:53)



So since the prefixes of length 2 are there, we will do the search two bits at a time. So, therefore we will form the trie. So at the first level of the trie, we will have branching based on two bits and not just based on one bit. So, let us see you know how do we organize this database. So we will have four branches based on 00 01 10 and 11. So 00 will correspond to p1 prefix. So, we will write here as p1 01 has p1 prefix 10 has p4 and 11 has p4. Now when you go to level 1, the next is prefix of length 4.

So now we will search the next two bits. We need search of next two bits. So, therefore will form the trie based on several search of next two bits, 00 01 10 11. So, this is p3 p3 and 00 will branch here, p2. Now this 10 11 00 middle here p2. Now this 10 11 10 01 is p5, 01. This is i think that this we have this routing database, we organized into this form. So, as you can see now is, let us that destination IP address comes whose you know let us say that this is the IP address. Now what we do is that, we will search first two bits 01. So, search two bits 01 will make a branch here. So, we come to p1. So we remember that match found so far is p1.

(Refer Slide Time: 42:14)



Now we look for the next two bits which is 11. So we will take a branch on this will come to p3. So p3 is the prefix which matches. So you can just see that if you have done if you have binary trie, bit by bit inspection. Here by taking multibit inspection just two memory accesses are required here and we can actually reduce the search you know search time considerably in fact as you can see here that since there are only three distinct prefix lengths. Either prefixes can be of length two or four or five you know your search complexity will be you know considerably reduced from earlier since prefixes could be of length1 2 3 4 up to 5, the prefix the search complexity of the order of 5 now it has been reduced to order of 3.

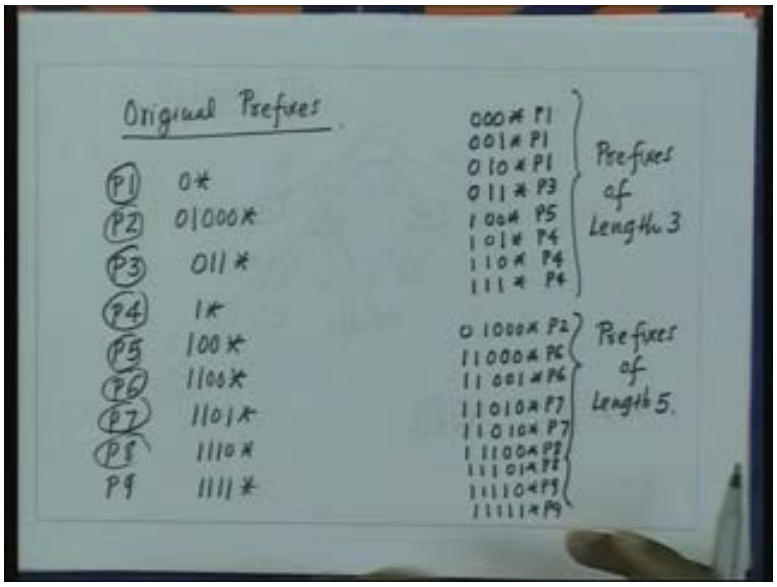
So that is how you know we have reduced the search complexity. Now of course, we need to process the routing database here in the sense that once the forwarding database once the forwarding database in the form of prefixes is there, we need to process these prefixes in such a manner that we need to expand the prefixes into fixed number of distinct prefix length. Now in which prefix length you need to expand is also a decision that one has to make and on that count also several algorithms have been proposed that you know how you can basically what are the best prefixes length for given database depending upon the statistics of the database but one more idea that i want you to tell that when you expand the prefix, there can be a problem that the expanded prefix can collide with the given prefix in the database and that happens then you have to retain not the expanding prefix, but the original prefix, just to illustrate you know that point i, i will take up an another i will take up the same example and let us say that our original prefixes. they are p1 0 star, p2 01 0 0 0 star p3 was 011 star p4 was1 star, p5 was1 0 0 star, p6 was11 0 0 star p seven is1101 star p8 is111 0 star.

Now let us say that we expand these set of prefixes into prefixes of length 3 and prefixes of length 5 only two. So, let we say we want do this into prefixes of length three and then you have do in prefixes of length five. So we will expand p1 first into prefixes of length 3. So when we expand this, we have 000 star which is p1, right then we have 0 01 star which is again the p1 we have 010 star which is p1.

Now when we go to 0 expanding it into 011, then you know that there is already a prefix which is p3. So, this has to be before p3. So, when i expand this p1 into 011, it collides with existing p3 so we retain the prefix p3. Now, we expand this p4 which is 1 0 0 star, the moment i expand this1 0 0 star it collides with the existing prefix p5. So i retain this to be p5. Now i then do as 101 star so this is like p4, then 110 star which is again p4 and111 star which is again you know i retain this to be p4, then you know so this how the prefixes have been expanded.

Now i expand the prefixes of length 5. So, first of all i like to take this p2 which is already a prefix of length five 01000 star which is p2 and now these four are the prefixes of length four which needs to be expanded. So let us take p6. So p6 is 11000 star so which is p 6 then1101 star which is again p6. So this is p6 is over, then p7 is11010 star which is p7 and11 01 0 star which is again you know p7 and then p8 which is 11100 star which is p8 and11101 star which is again p8 and of course we have p9 11111 star which is p9. So, in this case the original prefix database has been expanded into two sets of prefixes. Prefixes of length 3 and prefixes of length 5. So when organizes in the binary trie, what we will do is first you will search three and there will be how many branches as you can see there will1 2 3 4 5 6 7 8.

(Refer Slide Time: 49:46)



You know some of the branches and you will go to the, so you will first try to match you know three bits and if this three bits do not and depending upon first three bits match you will take a branch and then you will try to match the next two bits. So as the result as you can do as you can see that the search can be completed in in two memory accesses. First, you try to match the three bits and the next one you try to match the two bits.

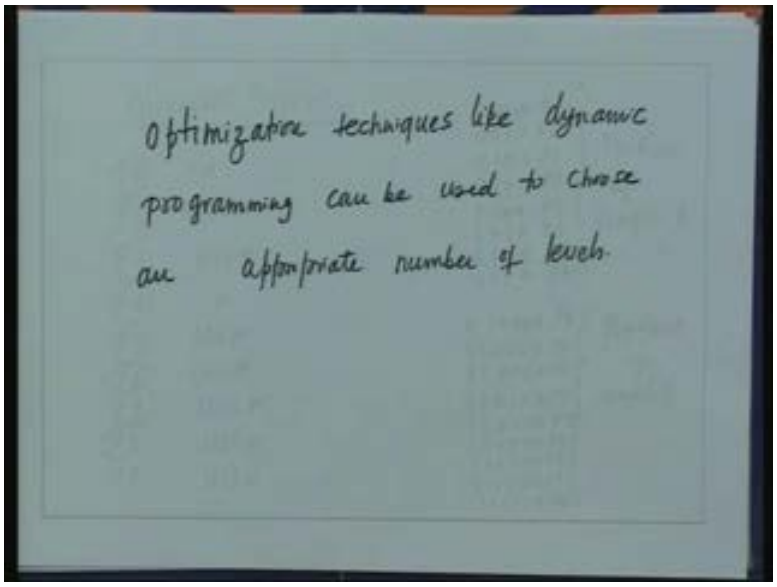
Of course, you know as you can see that you can expand all the prefixes into the length of 32 bit itself but what is the disadvantage? The disadvantage as you can see here that your size, the size the memory the memory size is increases otherwise you can expand all prefixes into the length of 32 bits and then you will require the memory size to be of order of 2 raised to power 32 then this will be equivalent to the exact match. So therefore, this multibit trie whenever you go for an

multibit trie this expanded trie we are actually increasing the size memory size that is required but you know reduce the search time.

So there is now the trade-off. The trade-off exists between the size of the memory and then the search time that we need to address. So therefore, you know one need to balance out the size of the memory that that can be useful and the search time. So therefore you know one need to look for the best prefixes. Therefore the cost associated with the size of memory and there is a cost of the associated with the search time. So, then this question is there that what are the best distinct prefixes prefix length or how can i expand those prefixes into the set of expanded prefixes such that i not only have an optimal memory size but also an optimal search time.

So therefore you know depending upon the statistics of the forwarding database this problem can be formulated as a dynamic programming problem and1 can actually deduce1 can actually deduce what are the best ways of dividing this set of prefixes into the distinct prefix. Now basically you know what I am saying is that, you can use optimization techniques. Optimization techniques like dynamic programming can be used to choose an appropriate number of levels.

(Refer Slide Time: 53:08)



So of course in the multibit trie, one of the problems that they are in is the update. The update is certainly more complicated than in the simple binary trie. So this good search time comes cost of little bit increase in the memory size and also a little bit more complex thing to add an entry or to delete an entry from the forwarding database, than the simple binary trie. So, now we will look at some more algorithms and techniques that can be used to not only improve the search time but also to reduce the size of the memory.

(Refer Slide Time: 53:57)