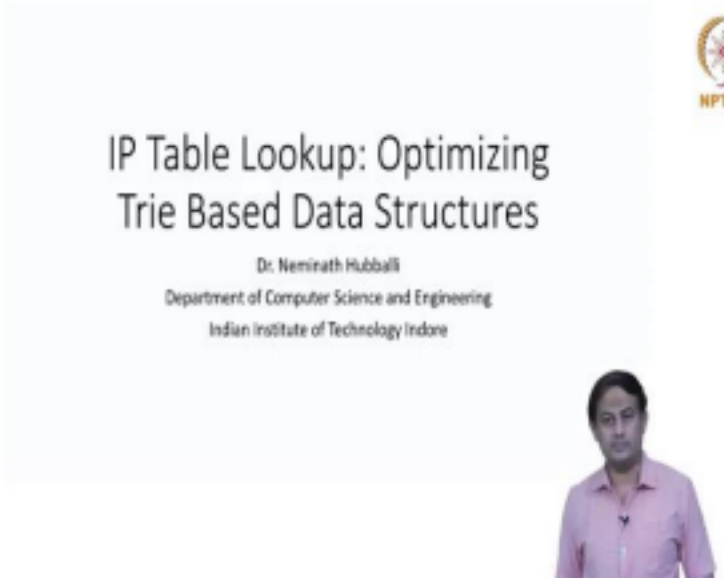
Advanced Computer Networks
Professor Doctor Neminath Hubballi
Department of Computer Science Engineering
Indian Institute of Technology, Indore

Lecture 7
IP Table Lookup- Optimized Trie based Data Structures - Part 1

(Refer Slide Time: 0:18)

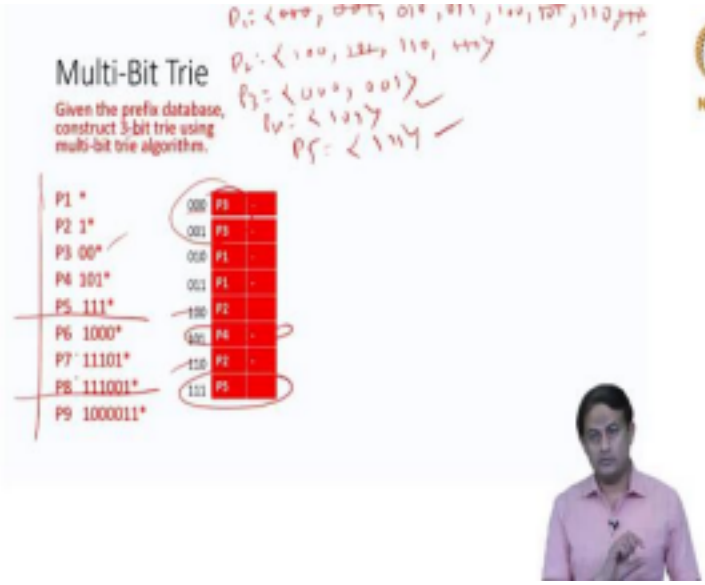


Welcome back, in the last lecture, we started our discussion on the IP Table Lookup, we saw some of the data structures available for doing the IP table lookup in the software, particularly, we saw binary trie and a couple of optimizations on the binary trie.

(Refer Slide Time: 0:39)

And we will continue that discussion today, we will see some more optimizations that are possible on the binary trie, and also we will look into the hardware-based type to lookup because software-based lookup, as we see it, is computationally expensive, several memory references are required and in order to speed up, hardware implementations are also sometimes used. So, the agenda is: in the first part will see the optimization of the binary trie is data structures and in the second part we will see the hardware-based implementation for doing the lookup.

(Refer Slide Time: 1:19)



So, we will begin by taking one more example of the construction of the multi-bit trie. So, multi-bit trie, we discussed in the last lecture, here is a set of prefixes that are given to us,

particularly, from P_1 to P_9 and he is asking to construct a 3-bit trie. So, meaning every time you do a comparison you compare 3 bits at a time.

So, as usual, there are 3 bits, then the table structure will have entries for 3 bits at a time and 2^3 is 8, so every table will have 8 entries in total. So, this is how the first level trie look like. The way again as usual the construction begins by looking at the prefixes which are up to length 3, so that is where the first level table will have entries, prefixes from the P_1 to P_5.

And these are how things look like, if you have an IP address starting with 000, then this is routed with the prefix P_3 which is the best match that you can find and 001 is also matched with P_3, and 010 is matched with P_1 and so forth. And as usual, the best way to look at is how exactly this is routed, what prefix actually takes precedence over the other one is to do the expansion and then cancel out whatever is the superset of the other thing.

So, P_1 actually has prefixes from all zeros to all ones up to 3 bits or 010, 011, 100, 101, 110, and 111, so this is the set of all these IP addresses which are starting with 000 or 001 and up to 111 are all matched with the prefix P_1. So, P_2 will match with anything that starts with 1, so 2 bits are variables, so those 2 bits can be either 00 or 01 or 10, or 11; these four prefixes are matched by these four IP addresses, which start with 100 and 111 or a lookup to 11 are matched with P_2 and P_3 will match 00 and the next bit can be either 0 or 1.

So, these two are matched with P_3 and P_4, actually is 101, P_5 is actually 3 bits, so we are expanding only up to 3 bits, so that is okay, P_5 is 111. So, anything if you get an IP address starting with 101, then P_4 will take precedence over the other cases, so wherever 101 is appearing, you will need to cancel it out; 101 is appearing here, 101 is appearing here, these two cancel out, and wherever you find the 111 those cases will also need to be canceled out because P_5 takes the preferences for the other one.

So, these two P_4 and P_5 are 3 bits, so those are given the preferences and then a 000 and 001 are given preference with P_3, so this 000 and 001 will cancel out from the P_1. One more thing is left P_2 also had 111, so that will also get canceled and the rest, I think, 010 and 011, so none of them are canceled out.

So, P_4 and P_5, P_4 are here and P_5 is here, so these two will take preference first you make an entry for P_4 and P_5, then you go to the P_3, that starts with 00, so 00 is here. So, these two entries are routed with the prefix P_3 and the P_2 100 which is here and 110 which is here, any IP address

starting with 100 and 110 is matched with P_2, and the rest of the entries are actually filled with P_1.

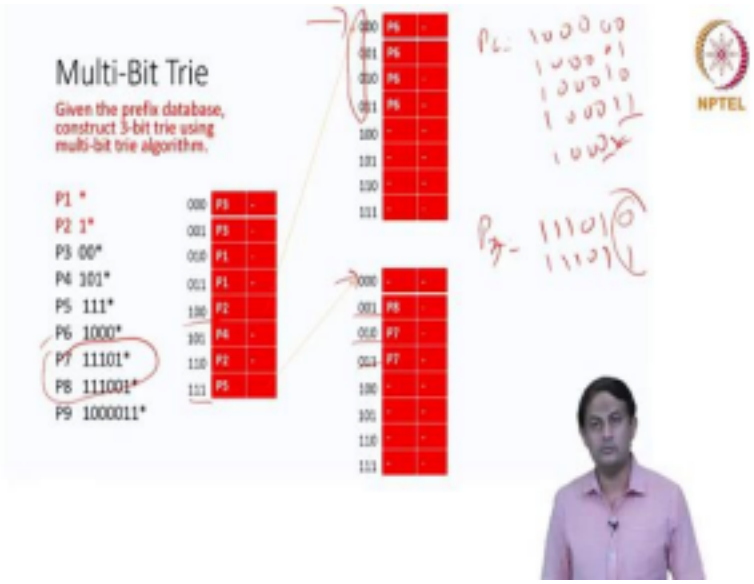
So, that is the default prefix to which any IP address is matched; that is how you figure out the first level; if you want to construct a K-bit trie, first expand up to the prefixes which are up to K bits or less than that and then whatever the prefix which is less than K bits you expand them.

And then if there are prefixes that are of the high preferences which are subsets basically, here in this case 3 bit trie, we wanted to construct any prefix which has 3 bits in one go, first, you give reference to them, and then the remaining you come to the next level 2-bit prefixes, 1-bit prefix and then finally the one which is the 0-bit prefix or the default prefix will take the preference. So, that is how you map the entries in the first-level table.

And now this will continue, so in the second level structure what we will do is, in the first level table 3 bits are matched and the next level table will also match another 3 bits. So, prefixes that are up to length six greater than three, but up to length six are now taken care of, they are going to appear in the second-level table. So, precisely P_6, P_7, P_8 are all in this range, so P_6 is four bits, P_7 is five bits and P_8 is 6 bits. So, these are the entries which are actually falling into the second-level table structure.

If any of the prefixes which are already marked or entered in the table level 1 structure are the supersets of these P_6 to $P_{8,}$ and it turns out that P_2 is a superset of all 3 of them P_6, P_7 and P_8, because P_2 is saying 1* so 1* will include 1000* also 111 01* also 111001* as well. And P_5 is a superset of P_7 and P_8, because P_5 is 111* and P_7 is 11101*, so we will draw the entries in the second level structure by taking this order of the preferences into account.

(Refer Slide Time: 8:45)



So, if you look at the structure, the second level structure would look something like this. Since, P_2 is a superset of P_6 1*, here in this case if it is a 111*, then we are going with the P_5 but only 1 followed by something, next level if it is 0 then we have to go with the P_6.

So, we will draw a pointer from this P_2 to the second level structure starting with the 000 and then 001 and something like that, and what we see is P_6 is 1 followed by 00 followed by 0*, so what it means is, if I expand this so P_6 will sound something like this 100 is fixed and then another 0 is fixed, but the next two bits can be either 00 or 01 or 10 or 11.

So, basically keeping these four bits intact 1000, and wearing the last two bits as per the convenience and then for all of these cases because all of them can be written as 1000*. So, 1000 is matched a t the first level and the next 0, this 0 would match at this level. So, wherever there is a 0, so in precisely in these four locations the fourth bit is 0 and I do not care really what is the next two bits for all those cases I write them as the P_6. So, all of them are getting routed by using the prefix P_6 that is the best match that you can find for any IP address starting with 1000.

Similarly, for the P_5, P_5 is a superset of P_7 and P_8, and as an expansion of P_5, a pointer from that in the first level table to the second level structure would have entries with the P_8 and P_7. So, that is what is shown in this table structure, in the second table.

So, P_7 is a 5-bit prefix and P_8 is a 6-bit prefix, so exactly one entry, so 111 is here and 001 in the second level is matched, 111 followed by 001 is P_8 and again, P_7 is 5-bit entry, you expand that, P_7 would include 111 followed by 01 and the next two one bit can be either 0 or 1, for these 2

cases 0 1. So, they wearing this one for these two cases where the next 6th bit is either 0 or 1 you mark those two entries with the P_7.

So, anything that starts with 111 followed by 010 or 011 will be routed with the port number, whatever the prefix P_7 is actually telling you. So, that is how the second-level structure is constructed. And so we are done with the case up to P_8.

(Refer Slide Time: 12:25)

Now, we need to look for the P_9, P_9 will appear in the third-level table and it is easy to note that P_9 is a subset of P_6, from the P_6 we will go to jump to the P_9. So, P_9 is 7 bits. We will need to expand P_9, make it 9 bits, so 1000011 are the seven bits and the two bits can be either 00 or 01 or 10 or 11, so keeping the first seven bits intact for all these four cases, I am going to construct these 4 prefixes which will be matched with the P_9, so, as an expansion of the P_6.

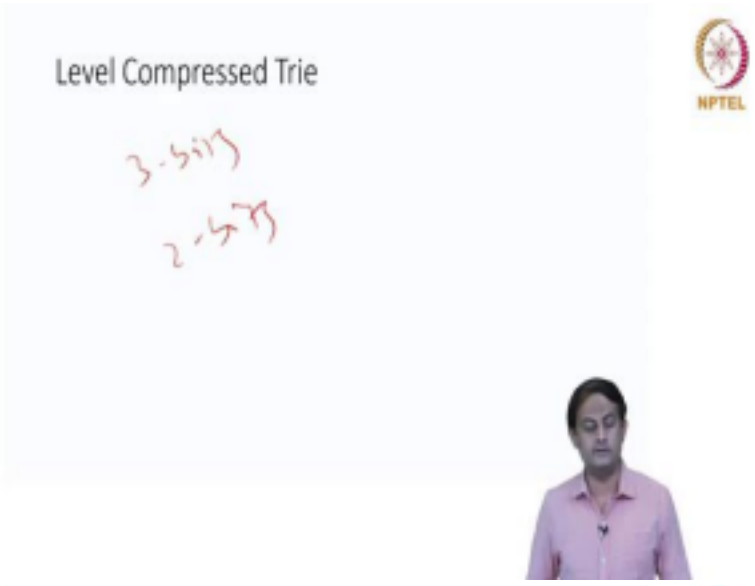
So, first 100 is matched here at the P_2 and next 001 is matched at this location, so this is the precise prefix P_6 and as an extension at the third level, you will have from P_6 to these 4 entries, what are those, 100, 101 and 110 and 111 these four cases are actually routed with the prefix P_9, whatever the port number P_9 is telling, using that it is actually routed to the next hop that is how the multi-bit trie is actually constructed.

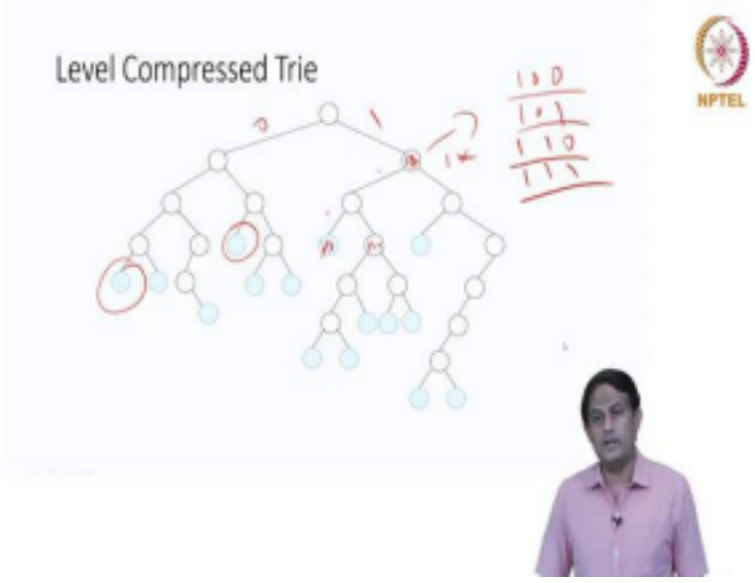
And just to summarize, what is the multiple trie zones you look for all the prefixes in your routing table and you decide what is K, the number of the bits that you want to match, and

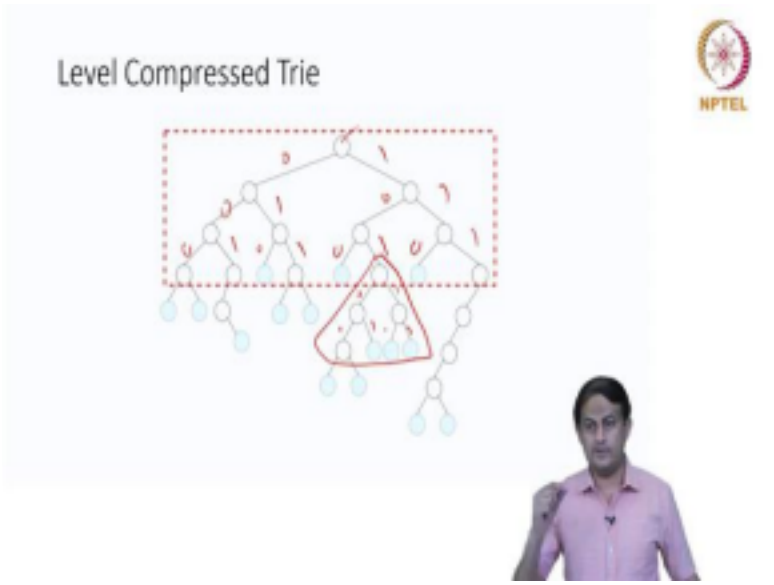
construct a table for K bits at a time i.e., first level table we have only entries for 2^K number of the possibilities and then the next level will have another 2^K number of the bits and in the next level another 2^K number of the bits will be matched something like this.

So, if your prefixes are not aligned to the exact match of multiple of the K bits that you have chosen then you need to expand that and fill the entries in the table. So, that is how multi-bit trie is actually constructed, but every table entry will match only the K bits.

(Refer Slide Time: 14:56)







So, that is one optimization that you can do on the binary trie, so if you have the ability to match a multiple numbers of the bits in one go, then you can do that. So, now the question is. Is it the only optimization that you can do on the binary trie? I have the ability to do multiple bit comparisons, but what we can see is some of the entries in this table structure, particularly at the second level and the third level, so these four entries are left blank and these four entries are also left blank and these four entries are also left blank.

So, in every table the entries in this sense in this table are the pointers and some of these pointers are not used and by virtue of that, you actually end up wasting some of the space in the table. So, it might happen that only one entry out of 2^K possibilities are actually having an entry

but the remaining ones are actually not having an entry. Then the remaining 2^K-1 number of the entries will be left empty, so that will end up wasting the space.

So, what the level compressed trie actually allows you to do is to have a variable number of the bits compression. So, meaning I can have a 1-level compression with the 3 bits, the second level compression can be with 2 bits and so forth. I can do 4 bits 3 bits 2 bits whatever is possible wherever the compression is possible that I am going to have in the structure.

So, let us try to understand how exactly this is done, so here is a binary trie which is actually having some nodes and an interesting thing to note about this binary trie is all the marking that is here, for example, this is colored with blue whatever the nodes are actually filled with blue color, it indicates that a prefix is matched at that node.

So, one interesting fact about this trie structure is all the markings are at the leaf nodes there is nothing that is at the intermediate node, for example, this node is not marked. So, similarly, several other nodes, including the root node, are not marked, and it is possible to convert any arbitrary trie structure where the marking is done at the intermediate level to a trie structure where the marking is only at the leaf node and the simple answer to this is how to do the conversion is to do the expansion.

So, for example, if this node is something that is indicating 1*, I can push this marking by doing an expansion of anything that is starting. So, because the left child is 0 and the right child is taking the path of 1, this K, if I mark this node that, indicates the prefix of 1*. So, I can make this a match with the 3 bits by doing an expansion and 100, 101, 110, and 111 for all these cases the nodes wherever these are best there I can do the marking. So, meaning, so 1* is not marked but 1 0 is marked, so this is node that is indicating that.

Similarly, 101 can be marked and brought it here so that all the markings in the trie structure can be pushed to the leaf node by doing such an expansion, so in effect, any arbitrary binary trie that you can construct for some prefixes for that there exists an equivalent trie structure where the marking is done at only the leaf node.

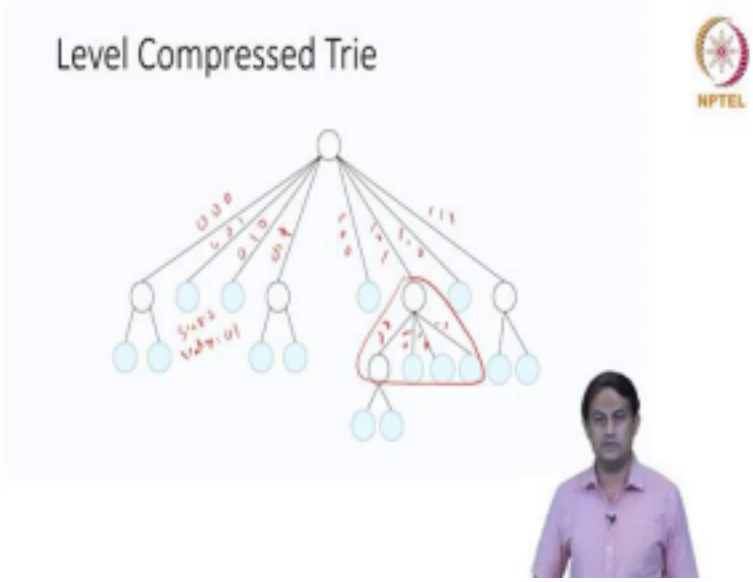
So, assuming you are given such a binary trie and what level compressed trie does is to it look

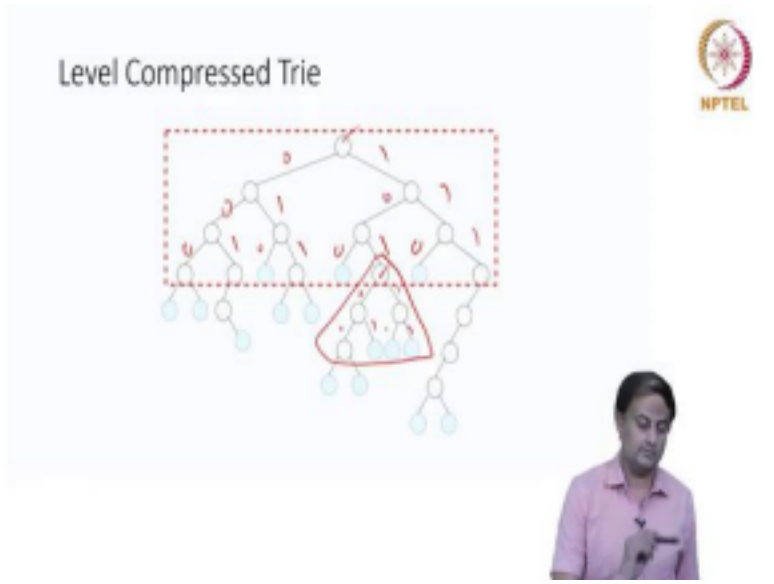
for opportunities to compress the variable number of such trie structures some bits and merge into whole. So, the way to go about it is, look for a complete subtree which is embedded within this particular trie structure.

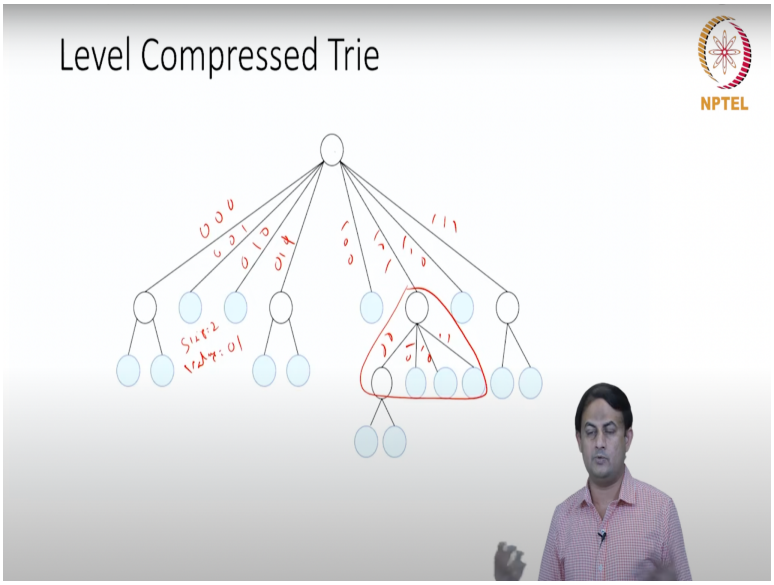
So, for example if you look at this diagram and all these nodes, so starting from the root node everybody has got two chances, this is 0 1, 0 1, 0 1, 0 1, 0 1, 0 1, 0 1, this is a complete binary sub tree which is embedded or which is a subtree of the original tree that we have. So, what will I do is? Wherever such complete subtrees are appearing, those portions, I will look into and then I will replace these entire subtrees as the direct descendants to the root node. So, in this diagram, this is one subtree and starting from this node, there is another substructure that is actually complete in nature. So, this is 0 1, 0 1, 0 1, so every node has got two chances, this is a subtree which is actually again a kind of the complete tree.

So, now the first level merging, I will collapse all these 3 bits and make a direct descendant of the root node with 3 bits, and the second level because this is also a complete subtree I will collapse this and make four descendants, for this particular node and then appropriately mark them.

(Refer Slide Time: 21:19)







So, here is the structure. So, this is the node which is indicating all zeros, left most part what was existing here 0 followed by 0 followed by 0 you are coming here, this is the node which is actually descendant of the root node, this is the node which is indicating 001 this is the node which indicates 010 and this is the node which is indicating 0 followed by 11 and this is the node 100, 101, 110 and 111, that is how the first level marking is done, this is as good as having a table with the 3-bit entries, 2^3 possibilities. And then, in addition to this I also collapsed this node, so this is the 00 this 01, 10 and 11, four possibilities because when you collapse a tree with a height of 2 then you will get these four possibilities.

In addition to that what I also do is I will bring the concept of the path compressed trie wherever

there is a node which is actually having a single path from that particular node, I am going to collapse all that and still remember what the skip length and what the skip value is. So, for example, this is the node which is actually 00 followed by 1, and I will push this into this level and then mark that this is 001 and what I skipped is of length two and the value that or the segment and that I skipped is a 0 followed by a 1.

So this is done at every level wherever there is such collapsing possible; so by doing the path compression I am going to re-label the nodes which are actually representing with collapsed entries, what is the skip length, and what the value is I am going to remember. So, you can think of a level compressed tree as a combination of what the multi-bit trie is offering you but in addition to multi-bit trie with a variable number of the bits at different levels and in addition to that, whatever the single chain paths are there those also we are collapsing and remembering what we have skipped and what the value that is skipped is. So, that is how the trie structure is compressed.

So, if you do this, then my height of the trie will also reduce at the same time, I will also be able to do multi-bit comparisons in one go, not only fixed K bits but a variable number of the bits as well. So, that is the concept of the level compressed trie. So, this goes with an assumption that the router has got the ability to do a variable numbers of the comparison, sometimes three, sometimes four, sometimes two, and also, the data structure is able to store what is the skip value and segment that is skipped.