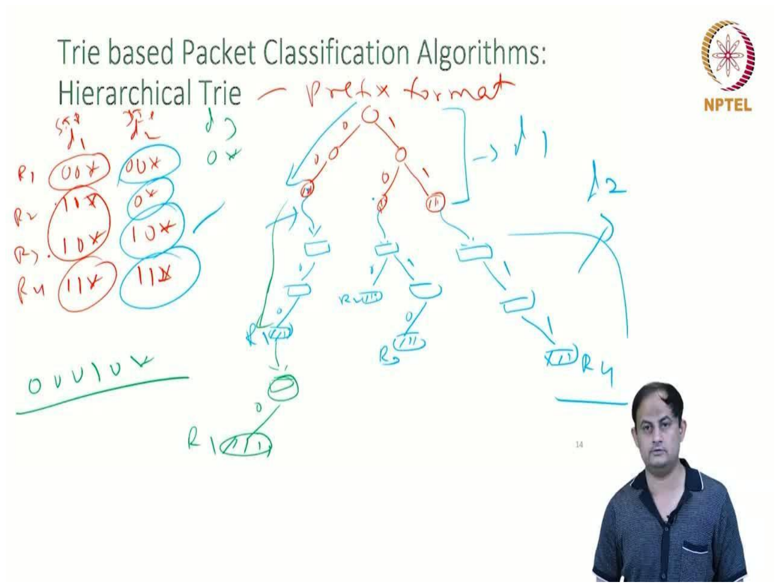
Advanced Computer Networks
Instructor Dr. Neminath Hubballi
Department of Computer Science Engineering
Indian Institute of Technology, Indore

Lecture 11
Packet Classification- Part 3

(Refer Slide Time: 00:18)



Trie based Packet Classification Algorithms:
Hierarchical Trie

So, with that background in mind, let us go and study the first data structure called the hierarchical trie. So, what we are doing right now is we understood that trie-based algorithms are software implementations that are going to work better alter for the lookup operation. Can I extend the same thing for the classifier as well? Classifier also, of course, has got set of rules and those rules are also in the prefix format.

Assuming all the fields are expressed in the prefix format. We can use this kind of trie based algorithm. So, what is the hierarchical trie we will try to understand with an example? So, let us say I got the rule with the two fields for the sake of simplicity, I am taking two here R1 has a prefix 00*, and the second one is also 00* and R2 has also got a set of prefixes is 10*, and this is 0*. You can simply think of this as the source IP address, and this is the destination IP address.

Assuming only these two fields, I want to build the classifier. So rule R3 says any prefix with 10*, and then the d2 also will have 10* and R4 says this is 11*, and then this is 11*. Now I want to build a kind of trie based data structure. And the way hierarchical trie works is you take one dimension at a time, and then construct a binary trie for one dimension, and then go to the second dimension, and then construct a trie for the second dimension.

So, the number of dimensions you have got those many times you need to do this construction operation, wherever it is applicable, you need to construct a binary trie in a hierarchical fashion. So first, let us take the case of the dimension d1 and then try to build the trie. And the rule R1 first, the source IP address field is actually saying 0 followed by 0, this is 0, this is 0. And this 00* is taken care of.

I am going to mark this to indicate this one. And the second possible prefix is 10* for rules R2 and R3. I am going to have a node here and then a 0, I am going to come back 1 followed by 0 and I am going to highlight this, these two are taken care of now. And the third possibility is 110*, 1 is already there and I am going to have a node here this will denote 11*. So, for all possible prefix combinations of the first dimension, the job is done now.

Now, what I am going to do is I am going to construct a second-level trie this kind of trie is for the dimension d1. And I am going to do construction for the second level. So, what are the unique combinations 00* is in rule R1. So, when this is applicable, it is applicable when R1 the first dimension is 00*. So, that is the chain from here to here. What I am going to do here is I am going to add a link, this is a no-cost link or you can think of it as a dummy link, and go to a node which is the root node for the second dimension in binary trie and construct the trie.

So, this is 00*, I am going to differentiate the first dimension, and second dimension, with the circular nodes for the first dimension and the rectangular nodes for the second dimension. So, this says that on 000* you go here and mark this as a new parameter. 00* for the first dimension you came here, and another 00* came here. This is the node which is representing the rule R1 in your classifier.

So, remember, this does not cost you anything you are not reading anything from the header. So, rule R1 is done now. And rule R2 is saying 1*. So, this is the node which is representing 10* And there are two combinations R2 says, as usual, I am going to have a dummy link or no cost link from this node to the next level root node and then on 0, it is something else, this is the node which is denoting or representing R2. This case is now taken care of.

And to go to rule R3, and for the same prefix 10*, this can be 10*. So, on 1, you come here, and then on 0, you go to the left, this is the node which is representing the rule number R3. So, 10* followed by 10* is the rule number R3 that is supposed to match. As usual, I add a no-cost link from this node to this node and then have two nodes one on the right and the second one on the right, this is the 1. So, this is the rule, which is representing rule number R3.

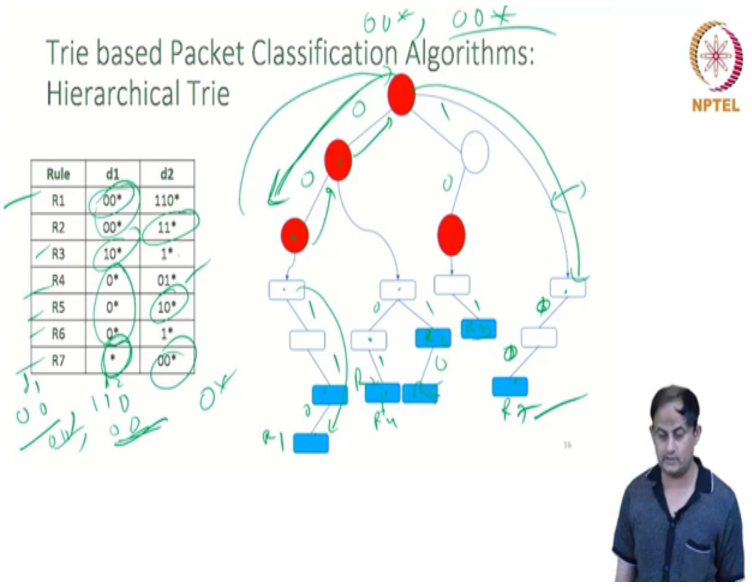
So, this could from here to here, whatever you have, this is the dimension d2, and this is the set of nodes inside your trie which is corresponding to the dimension d2. So, like this, I can go to any number, if I have a third dimension, all that I need to do is again.

So, let us say I have dimension d3. And then the combination is something like this, maybe the dimension d3. If it is a prefix format let us say this is 0*. And what I need to do is go to R1. So, this is 00* is taken care, and another 00* is taken care of. And now add a no-cost link to the third level this one, maybe I will add oval shape this one and then 0* go here, and this will become your node, rule number R1 so, I need to highlight this. So like this, any number of the fields that you have got inside your rules, that is the hierarchy in the order first from the d1 to d2, d3 and so on I keep on proceeding. So, this kind of structure is called the hierarchical trie.

So, now it is easy to note that. So, if the fields are some W-bit width, so in the first level, you might have W pointer references. And if the second dimension is also W-bit field, then you will have again, the W number of the pointer references to do and so forth. So if W is the highest width, when I say width this is the width 00010*. So, what is the width of this, in this case, is the 5-bit, w is the width, and d is the dimension. In the worst case, what is the lookup time this data structure is going to take W times the d.

So, for every dimension, I am going to make w references, then overall, to figure out what is the reference is going to take, one rule evaluation is going to take W x d. And it is possible that whatever rule you are going to evaluate may not work out, you need to backtrack. Again remember, one rule, one track evaluation if I want to evaluate one rule, then that will take W x d amount of the number of the pointer references. When it fails then we will come back to that in a minute when we discuss the performance of this algorithm at the end of or after taking a couple of more examples. So, this is how you construct the hierarchical trie, a binary trie for every dimension attached appropriately inside this structure.

(Refer Slide Time: 9:19)



Trie based Packet Classification Algorithms: Hierarchical Trie

Rule	d1	d2
R1	00*	110*
R2	00*	11*
R3	10*	1*
R4	0*	01*
R5	0*	10*
R6	0*	1*
R7	*	00*

So, let us take a second example here is a classifier with the seven rules, and there are only two dimensions just for the sake of simplicity, R1 is 00* and then 110*. And let us try to construct the hierarchical trie for this classifier. So, as usual, I take one dimension and then build a binary trie, and 00* is the one I want to build. So this is a 0, this is a 0 and this 00* is now taken care of. And similarly, another possible case is you can have 0* as this one that is why I have highlighted this node also with the red color and 00* is given by this.

And what is the other combination you have got, the other combination is 10* which is in the rule number R3 this is 1 this is 0 and this node is highlighted. Is that the only thing that you got and * is that, 00*, these two are taken care 10* is taken care 0* is taken care and * is taken care because this is the node which is indicating the * the root node itself.

So, now let us go to the second level what rule R1 says when the prefix in the dimension d1 is 00*, we come here and 110* is what you will need to look for or 11* this is the rule R2, 1 and 1 this is the node which is indicating rule number R2 and 110* is the one which indicates the rule number R1. This is the node that is actually getting the rule number R1. So, R1, and R2 story is done now.

And now, with the 0*, I came here to this node, I had a pointer to the level, and what are the possible combinations one says the 01* this is a 0 and this 1, and this is the node which is actually corresponding to rule number R4. Yeah, and then 10* is also in the combination; this is a 1, this is a 0, and this is a node that actually indicates the rule number R5.

And rule number R6 says 0* followed by 1* this is the node that is actually indicating the rule number R6, and I continue doing this. 10* is at the first level R3 case is left and then in 1*, so this is the node which is actually indicating rule number R3, and I continue doing this.

So, there is rule number R7, which is actually *, * is this node I took in no cost link to this node, the particular root of node, and then this says 00*, 11 you come here this is the node which is actually indicating rule number R7. So, R1, R2, R3, R4, R5, R6, R7, all the seven rules are taken care of. That is how you construct the hierarchical trie for any given classifier or a set of rules.

Now, let us try to evaluate. So, let us say my input IP address is 00 for the dimension d1 and then for the dimension d2 it is 110 then for these two 0s, I take this path, for this 110 I take this path, and I see that my current standing is at this particular node, and rule R1 has matched.

So, let us take the second case if let us say the dimension d1 is still 00 and the second case dimension d2 is also 00. Now, how does the rule evaluation starts I read the first 00 and as usual, I take this path and come to this particular node. And then next to 0, I read but I do not have for the second level, I come here with the no-cost link once I come here, I come to this particular node and then there is no node that is actually connected to this particular node with the possible input of 0 then what do I do.

So, in order to do that, what you need to do is you need to go back to this particular node or at one level to this particular node and then try other combinations. So, there is a combination if 00* is not the thing, then 0* might work because there are some rules which are in the first dimension which are having the 0 starting with 0*, 00* is a subset of 0* I go to the one level higher and then try to evaluate.

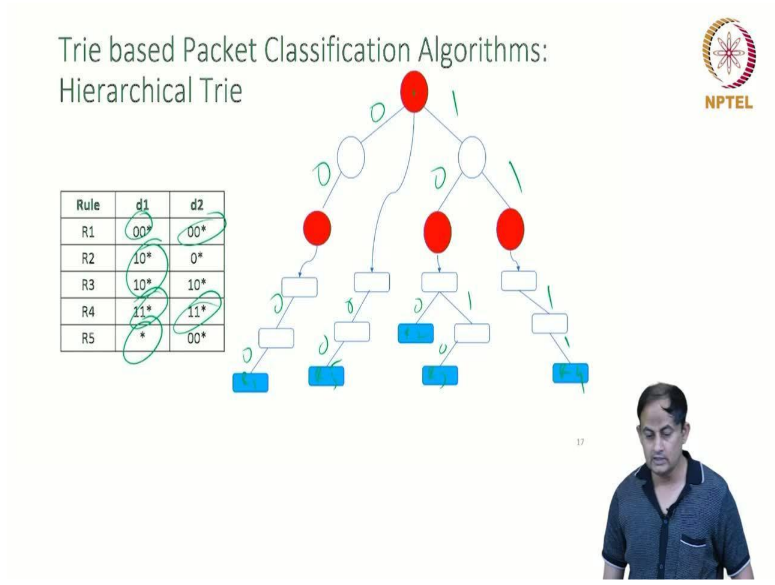
So, if I come to this node, then again I take the no-cost link to this particular node and try 00* in the second dimension again, there is no link I come here, and then take this. I come here and then there is no link or the node connected to this particular node which is having another possible input of 0, so that fails.

Again, I am going to backtrack all the way to this node and again go back to the root node. And then what is the next possible thing that I can do 00*, is it a subset of any rule other than 0*. It happens that * is actually a superset of 00* or in other words, 00* is the subset of 0*. I try a combination with the *; maybe I will take this path and come to this node. And then I try 00*, so, in this case, I wrote wrongly think this is supposed to be 0, this is supposed to be 0, just make a note that there is a typo of I put the nodes correctly, but wrote 1 in the label that is supposed to be 0.

So, 00 in this dimension, I try to match and I find a match, and I see that R7 is the best match. So, remember, again, if your input is 00* for the first dimension and 00* for the second dimension.

Then you first attempt R1; it fails; the second time you attempt R2, it fails. The third time you try with R4, and that fails with R5; it fails with R6, it fails. And finally, we need to backtrack to the root node when R7 is * that time 00* is in the second dimension; we will come to the rule R7. So 00* and again, 00* for the d1 and d2 will take you to the applicable rule R7. That is how you find out what is the applicable rule for a given set of rules and the input or given classifier and the input packet.

(Refer Slide Time: 16:56)



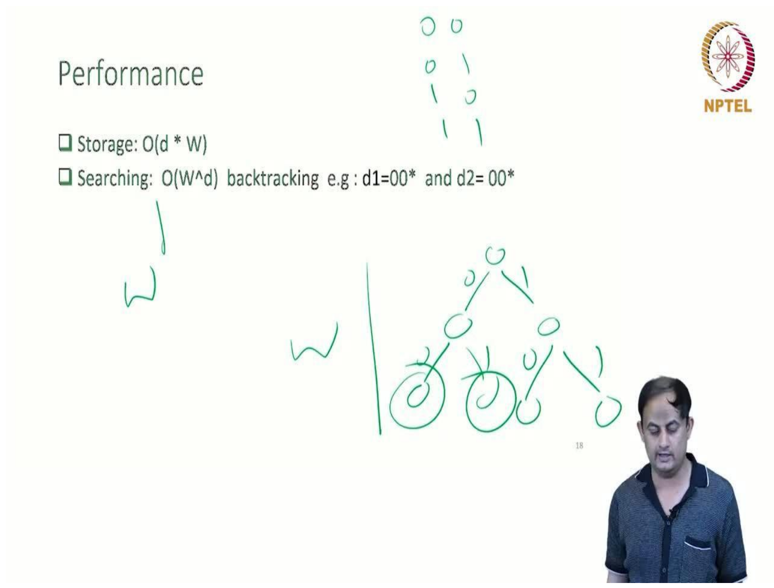
Trie based Packet Classification Algorithms: Hierarchical Trie

Rule	d1	d2
R1	00*	00*
R2	10*	0*
R3	10*	10*
R4	11*	11*
R5	*	00*

Let us take another example and try to construct the same hierarchical trie. Here is the second example, so the rule R1 says 00*, so 00* you come here. And second, this is taken care and for these two, I am going to add 1 followed by 0, this is taken care and the third one is 11* and this is the node that is created.

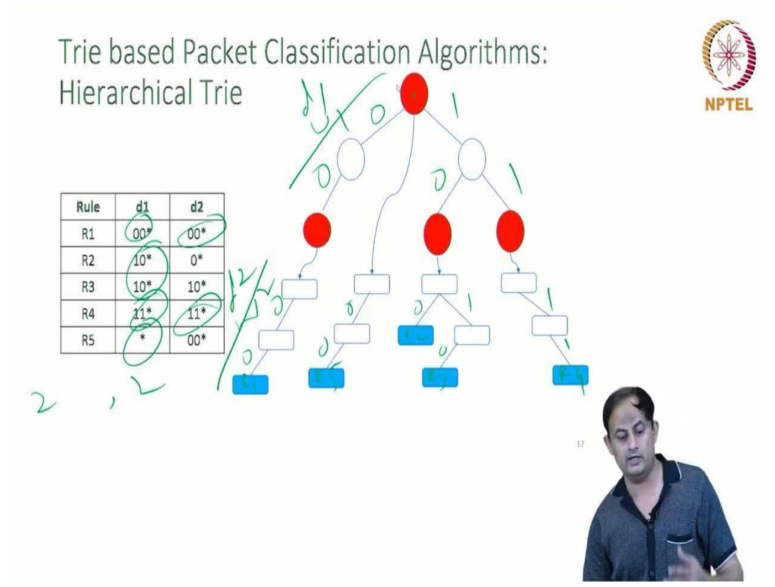
So, this is taken care of and * is taken care of by making the root node itself or highlighting the root node. So, one dimension is complete, and the second dimension: Take the first rule which is 00* this is the rule R1. So, this is 0; this is 0 and 1 for 10 combinations. So this is the 0 for this one, this is the rule R2 and the second is 10*, this is the rule R3 and the 11* in the first dimension followed by 11*, so this is the rule R4 and finally for a * a 00 *, so this is the rule for the R5. So, this is how you construct the hierarchical trie, one dimension at a time.

(Refer Slide Time: 18:24)



Performance

- Storage: O(d * W)
- Searching: O(W^d) backtracking e.g : d1=00* and d2= 00*



Trie based Packet Classification Algorithms: Hierarchical Trie

Rule	d1	d2
R1	00*	00*
R2	10*	0*
R3	10*	10*
R4	11*	11*
R5	*	00*

So, with that background, let us try to understand the performance of this kind of the hierarchical trie in terms of storage it requires d x W, W is the width d is the dimension. So, in every dimension W number of the nodes or combinations are actually created.

So, when I say d x W, it means that for all possible combinations of W, meaning if W is 2 bits you need to take care of these many combinations (four), so my tree would look like this one 01 and 01, 01 so, this is the W width trie that much storage is required. This I need to replicate d number of the times dimension 1 this one and for every node here, there is another W bit trie that is possible i.e., a full tree is possible.

So, W bit trie complete binary tree for one dimension, another W bit complete binary tree for a second dimension. Totally how many number of times you need to do this, d times as many number of the dimension, that is the amount of the storage that this structure would take.

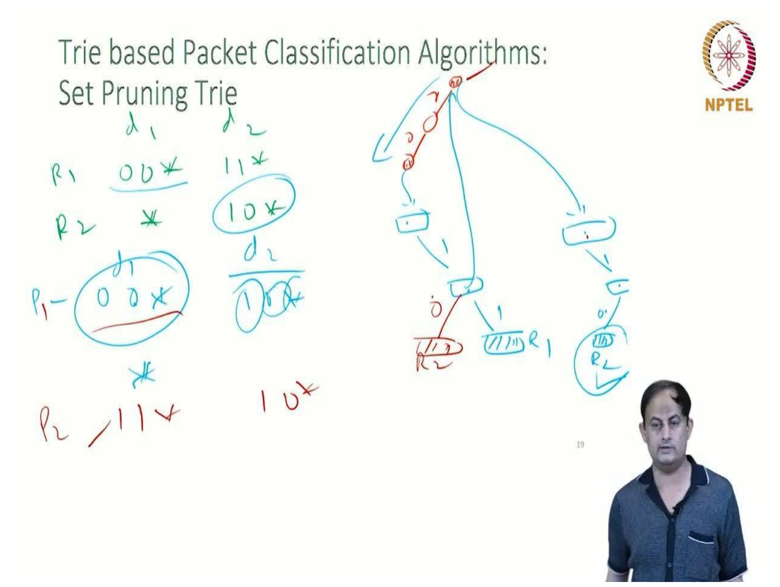
And the searching time, in the worst case, one branch searching from the root node to one of the branch searching would take d x W this is the d1 and this is my d2, and width here is W1 is 2 and W2 is also 2 and you require four number of (d x W number) of the pointer references for one thread. And if the prefix match fails, for this case, one rule, you need to backtrack and then explore the other possibilities.

So, in that case, what is the worst case number or possibilities of the backtracking that I can do, and in total, how many numbers of references of backtracking I need to try is W number of the back references one dimension I go and then I try the alternatives. So, W^d, W is the one dimension backtracking and how many times I need to do d number of the times.

So, I need to in the worst case, I need to explore all possible complete binary trees in every dimension that turns out to be W x W x W.. how many times d times. So, those many times if I explore all combinations, then we will get the worst-case possibility that is what the search time for this classifier is, W^d.

So, now, is it the best I can do? What is that alternative thing that I can do W^d is actually quite complex; many, many numbers of backtrackings we need to do in the worst case. Can I minimize the number of backtracking? And then, if I want to minimize the backtracking, what is the alternate modification that I need to do inside my hierarchical trie or this data structure, or how do I actually do that?

(Refer Slide Time: 22:04)


Trie based Packet Classification Algorithms:
Set Pruning Trie

The answer turns out to be a modification to the basic hierarchical trie. This is called a set pruning trie. What is done here is for any combination of the rule set if the evaluation of one rule fails, and if instead of doing the backtracking, I will explore what are the other possible applicable rules for this currently where I stand and then attach those combinations of nodes as a subset rule, let us take an example and then try to understand what I am trying to say.

Let us say rule R1, you have got two dimension d1 and d2. Rule R1 says the first dimension is 00* and the second dimension is 11*and rule R2 says this is * and then this is 10 *. Now construction of the trie, as usual, starts with one dimension at a time. Now let me construct the hierarchical trie for this one and then try to do the modification.

So, 00* will take me here and then I will highlight this, and the second is * itself. And I will highlight this node. This is the first dimension d1, all combinations are taken care. And now I go to the second level and then try to build this one so, in here add a new cost link to the second and then the I take 11* is a combination I add a node here this is on 1 this is on the 1 and I highlight this. This is the rule number R1.

The second one says when the d1's prefix is * that time on a 10 combination, I am going to match the rule R2. So, I will add a no-cost link to this one. And then 1 on the right-hand side and 0 on the left-hand side, and this is the node which is actually denoting rule number R2. Now let us say the packet has got this combination d1 has got 00* it might be more than two bits, that is okay. And the d2 fields start with 10*. Now, I take these two bits and I traverse this link, and then with no cost link, I come to this node and then try to read 10 *.

So, 1 is there I come to this node and then try to read 0, it is not possible because there is no left child to this particular node right now so that is not possible. And I need to backtrack, backtrack where? So, what are the other possibilities? What is the node if 00* is not applicable? What is the other rule that is applicable here in this case of the star from here all the way to this start node I am going to backtrack.

So, then I will try to explore because 00* is a subset of *, this rule R2 case is valid. Now, I try to read 10*. So, I come to this node on 1, I come here on the dimension d2 this one and then on the 0, I come to this node that is matched, now I say that for 00* 10* rule R2 is the actually applicable one. This is the best match that I can find.

Now, this requires backtracking all the way from this node to the root node I went. And then, if, let us say, I do not want to do the backtracking, backtracking is the one that is actually adding complexity to my hierarchical trie which is why this runtime the lookup operation is becoming expensive. What I want to do is I take this rule sets, and then 10* is another possibility that is applicable when 00* is also a prefix.

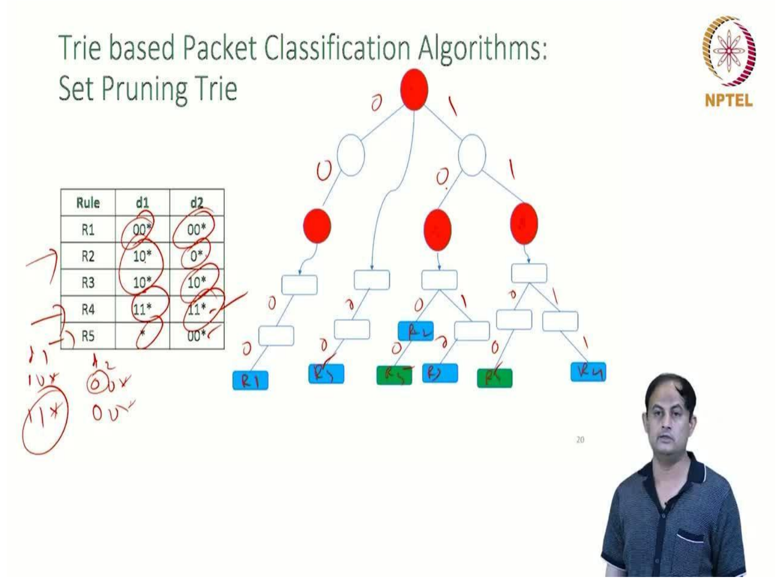
So, I take that combination and then add that corresponding nodes to this itself. So, let me highlight that with a different color, this is the 0 when your input is 00* as is the case in this one, and 1 followed by 0 is valid, but it is valid it is taking me to say that this is the rule number R2. By replicating when the input is, let us say possibility if packet let us say P1 and P2, P2 has got 11* and then 10*.

So, in that case, evaluation of this one anyway, I do not have any child with the value 1. So, the only possibility is to treat this as a * and then go to this node and then do the evaluation, 10* is matching. But even when the 00* is there, that time also because of the second dimension evaluation fails, I need to evaluate or explore the rule number R2; rule number R2 is applicable when the dimension d1's first rule 00* is also a possibility that is why I replicated here.

So, this avoids by taking such applicable rules when 1 is the one rule, prefix is a subset of the other one, such overlap is possible. I am going to do the node replication and then appropriate branches to the existing structure and then try to avoid the backtracking, thereby bringing the efficiency or the number of the pointer lookups are actually minimized.

But it is obvious to note that comes at the expense of the node replication. Node replication implies that you require more storage to store your classifier, to keep it in the router the memory; overall memory size requirement is going to increase. So, this kind of structure which comes at the cost of the or the expense of the storage space, but avoids the backtracking is called the set pruning trie.

(Refer Slide Time: 28:41)



Trie based Packet Classification Algorithms: Set Pruning Trie

Rule	d1	d2
R1	00*	00*
R2	10*	0*
R3	10*	10*
R4	11*	11*
R5	*	00*

Let us take a couple of more examples. So, here is a classifier. And it says that you construct the set pruning trie. So, let us again do one dimension at a time. As usual, let us first construct the hierarchical trie 00*, for this combination, I have got 00* and then a * is also possible that is also the root node which is highlighted these two prefixes are now taken care of. 10* is another possibility this is a 1, this is a 0, this is a node which is corresponding to 10* in the dimension d1 and 11* is another possibility, this is done.

Now all combinations in the first dimension are taken care. Now, as usual, let us go here. So 00* is another possibility 10 and 0 and this is the node corresponding to the rule number R1. With R1 again I continue with this 110* is there, I come here and then with the 0 this is the rule number R2 and then this is taken care this 10* is 1 followed by 0 this is the node corresponding to rule number R3. And again, 11,* and then this is a 1, this is a 1 and then this is the node which corresponds to rule number R4.

Now, at this stage, we have constructed trie for all the combinations; one more is left out this is combination this is the node which corresponds to the rule number R5 * and then a 0 and a 0. There is a little repetition in the order in which they are appearing. Now, you need to add the replicated rules; the nodes which are actually highlighted in green color are the nodes that are replicated.

So, what are the rules that are replicated? So, when the input in the first dimension is 10* you come here. And then, if your subsequent input is 0, you come here R2. And if that fails, let us say the input is this one 10* in the dimension d1, and d2 is 00*. So, 10* I come here, and then reading the first bit of the dimension d2 I come here, I am trying to explore whether 10* and this rule number R2 is applicable.

And I read the second 0; then I have 00*, 0* is actually a subset of the 00*. But in that case, instead of saying that this is the rule number, R5 is more specific than R2. And what I am going to do is, I say that instead of doing the backtracking, I am going to say that rule R5 is applicable here. That is why I replicated this rule at this point of time.

Similarly, if the input is 11 and the * in the first dimension, and then 00* in the second dimension, so 110*, I come to this node, and then on a 0 and a 0, I come to this node. So, 11*, I am looking for 11 and when I am reading the 11* of the first dimension, I am trying to evaluate the possibility of the applicable rule R5. If that fails, then the second time, I was supposed to get a 11*, but that is not happening. So if that is not the case, I am going to try another rule which is the possibility of R5.

So, which is 00*, so this is going to be R5, even if the prefix in the first dimension is 11*. So 11* is a subset of the *. And in the second dimension, I did not get 11*, but then I need to backtrack in general instead of doing that, I also say that 00* is applicable right here. So the by doing the right replication, I come here. So, that is how I construct whatever is the applicable rule for that combination. I am going to add the replicated rules and then construct the set prune trie.