


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


Lecture 12
Packet Classification Implementation
Part 1

(Refer Slide Time: 00:17)




Packet Classification

Dr. Neminath Hubballi
Department of CSE, IIT Indore




Welcome back. In the last lecture, we started our discussion on the topic called packet classification. Just to refresh your memory, packet classification is done with a set of the rules. And the rules are written using the different attributes, be it the source IP address, destination IP address, and the transport layer protocol fields, and then the port numbers or any other additional fields that you want to add in. So, we will continue the discussion on the packet classification today as well.

(Refer Slide Time: 00:53)



Overview

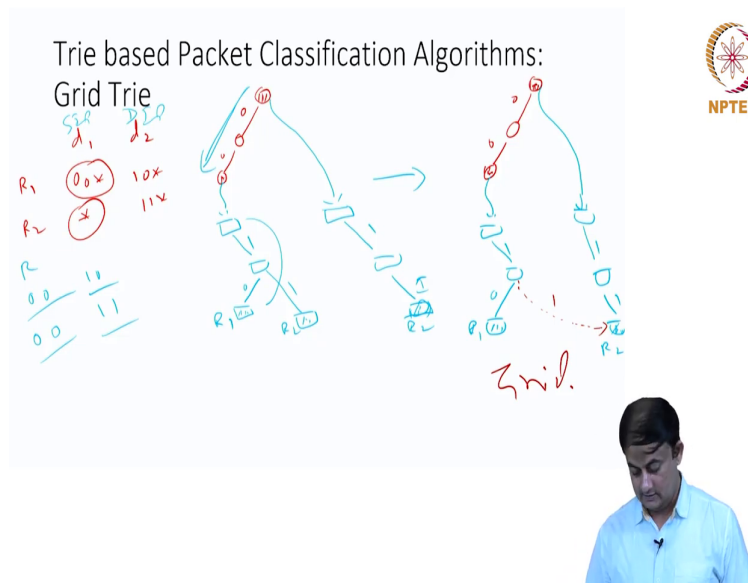
- Optimization on Hierarchical Trie
 - Grid Trie
- Extended Two Dimensional Trie
- Field Level Trie
- Geometric Interpretation of Packet Classifier Rules
- Heuristic Method for Packet Classification
- TCAM based Packet Classification
 - Handling Range Expansion



And the agenda for today's lecture goes something like this; we saw some of the data structures for implementing the rules of the classifier. And we saw one optimization that we could bring, particularly something called the set prune trie on the basic hierarchical trie structure. And we will see one more optimization on that one, today. And then, we will see some of the heuristic methods to do the classification.

And we will also look into something called the field-level trie, which is a different kind of trie for doing packet classification. And then, at the end, we look into how we implement the packet classifier using TCAM memory as well. So, that is the whole agenda for today's lecture.

(Refer Slide Time: 01:54)



A data structure called the grid trie is an optimization on the hierarchical trie, and even the set-prune trie as well. So, let us see what the idea behind the grid trie is. And in order to understand that, let us take a simple example, here is the classifier with two rules with the two dimensions d1 and d2.

And rule R1 says the d1 is 00* and then d2 is 10*. And rule R2 says this is * and then 11*. So, these are the two rules in my classifier on the two dimensions. And first, let us construct the set-prune trie and then ask, what is the further optimization that we bring here? So, as usual, I go one-one dimension at a time. So, I will start with the root node, and then on a 0, you come to the left, and on a 0 come to the left; this 00* is handled.

I will mark this as with the highlight, and then you go to the * that is done by highlighting this node itself. So, both the prefixes of the dimension d1 are now taken care of. Now, let us go to the second dimension. So, d2's first rule R1 says you need to have the path for 10* and that can be done by having a no-cost link to the left node then one a 1, you come to the right, and then on a 0, you go to the left, this is your rule R1.

And rule R2 says this is *, so I just add a no-cost link from the root node to this node directly. And then it says on 11, so this is a first 1 and this is the second 1. So, this is your rule R2. And now what we said is, in the set-prune trie, and if you happen to take, let us say you get the input as a packet, assuming d1 is the source IP address and d2 is the destination IP address.

And this packet has got to the source IP addresses starting with a prefix 00, and the destination IP address is, let us say 10*, then on this 00 I traverse this path, it is well and good, on this 10 I traverse this path and then reach the node with the label R1. So, that is fine. So, we say that this is classified with the rule R1. But on the other hand, if you get input 00* and then 11*, what was happening is on the first 00, you traverse the node and then on this 11, we do not find a path here, so we are going all the back to the root node and then trying the alternative path. Instead of doing that, what we said is you add this node here itself and then label this with the rule R2. And so, by that you just avoid the backtracking. So, that was the resulted set prune. So, this set prune trie was giving you optimization with the, at the cost of the node replication.

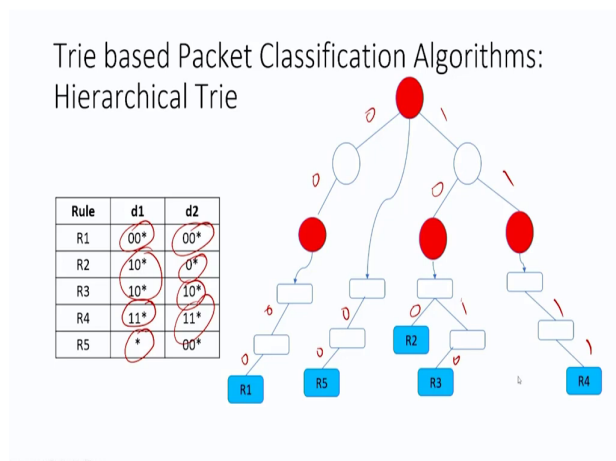
A simple question that we ask here is why to replicate the nodes. So, instead of replicating the nodes, anyway, a node with the label R2 exists here in the trie, can I add a pointer to that node? Wherever I was doing replication, just add that to a pointer to that node? Although it may not give you a very big reduction in the storage space and the number structure of the trie. Nevertheless, anything that you can avoid is actually a good optimization.

So, that is precisely what we do here in the grid trie. So, what we do is, instead of replicating the node, we actually add the pointer to the respective node itself. And thereby, we actually try to avoid node replication. And so, this is how the grid trie looks like. On row 00, the structure is there. And this is highlighted, this is highlighted. And what I do is add the no-cost link from here to this one, and no-cost link from here to this one, and on a 1 and a 0, this is anyway representing R1, I retain that.

And then for this, I require 1 and 1, I have this, and label this with R2. And whatever the additional node that I was adding that I am going to avoid by adding a link from here to this node itself, through this on 1. So, this is what the trie structure now looks like. And this we call the grid trie. So, it is an optimization over the set prune trie, in terms of the number of nodes that exist inside the trie structure.

So, that was the thing. So, we will take a couple of more examples and see how much reductions, specifically, it can bring in terms of the number of the nodes and although as I said, it is not a very big reduction in the number of the nodes, nevertheless, there is some optimization.

(Refer Slide Time: 08:02)



So, here is a classifier with five rules R1 to R5. And the values or the prefix values for the dimension d1 and d2 are shown here. And as usual, we go one level at a time, one dimension at a time, and construct the trie. So, here is the node, so this represents the *, the root node itself is highlighted, so that is okay. And this one is 00, so this prefix is now taken care of. And I continue.

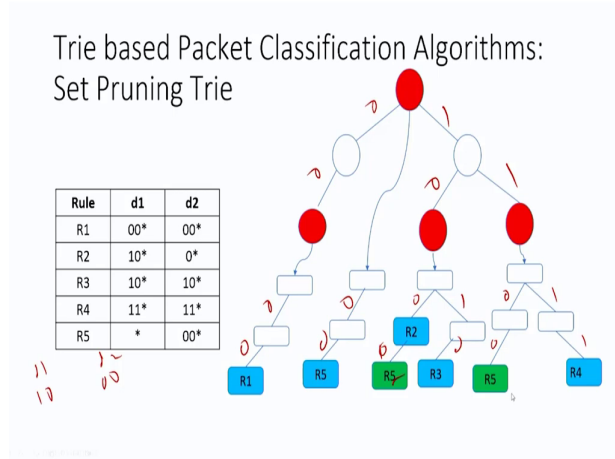
So, 10* is another unique prefix that you find in the first dimension, these two are taken care. And there is a third unique prefix, which is 11*, which is here. So, the first level trie structure is now done. So, all the unique prefixes of the dimensional d1 are now taken care of. So, as usual, we move to the second dimension and then keep adding the nodes as a descendant to the nodes.

So, what are the unique prefixes? 00* is another unique prefix, but that is applicable when the first level prefix is 00*. I add a couple of nodes as a descendant to this node using a different kind of node structure; here is the rectangle, and I label it as R1. And another prefix on 10, the second level is 00* so I am going to have it here. And 10* with the prefix 10* is also there so I am going to add two nodes here, 1 and 0; this I label as R3.

And similarly, with 11* and 11*, I am going to add two nodes here. And then for the *, I am going to have 00 here. So, this is the hierarchical trie structure for this, this set of the rules in

the classifier. So, if I convert this into a set prune trie, as usual, I am going to replicate the nodes instead of backtracking.

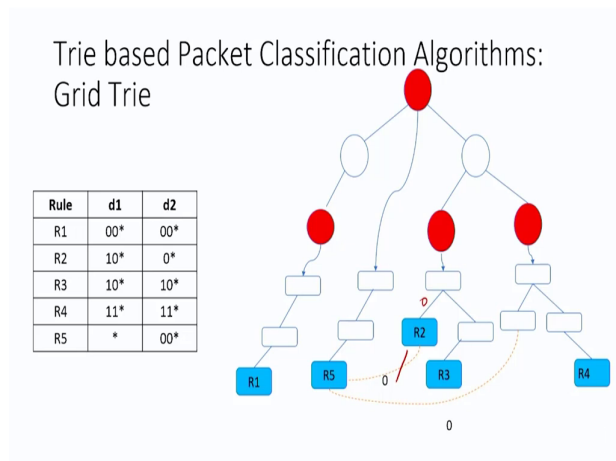
(Refer Slide Time: 10:15)



So, how does the set prune trie look like. So, first level nodes are, as usual, they are there. And the second level nodes are also as usual, whatever they are in the hierarchical trie, all of them are here as well. So, this is what the hierarchical trie was. And what I am going to replicate now, the nodes which are actually to avoid the backtracking. So, if let us say, on the rule, 10 star in the first case, and then a 00 star in the second case, would take you to the normal, this is 0 0, this is 0 0 0 0 1 0 0 1 1 0, this is 1 0 1, and then this is a 1.

So, the green color nodes are the ones which are replicated. So, if you happen to read 10 in the first place, and then a 01 in second place instead of that one, so what we are saying is, you will need to sorry, this is 0, 00 star. So, if your input packet contains the prefix 10 for the dimension d1, and 00 for the dimension d2, then the rule that is applicable according to these set of rules says, basically R5, that is why I replicated this node. And then I continue doing this. So, on 11 star in the first dimension and 00 star in the second dimension will also take me to the rule R5. And these are the two rules that are replicable.

(Refer Slide Time: 11:54)



So, if you convert this into the grid trie how the trie structure will look like? So, as usual, the first level nodes are there for dimension d1, and these are the four nodes that are highlighted corresponding to each of the unique prefixes in dimension d1, and I add the nodes for the second level as well. So, this is how the trie structure looked like. And I was replicating on the left-hand side node, R5 was there.

And instead of doing that, what I am doing is I am adding a link from this node R2 on the input 0 to this R5 itself. And there is another node here, which was actually descendant of this node, which is going with the label 0 on the left-hand side, that is also going to node R5. So, thereby, adding these two links, basically, I avoided replication of the two nodes in the trie structure. So, that is what the grid trie is all about.

that is the rule R5 because, again, 0^* is a superset of the 00^* . So, that is why we are replicating here. And then the, if you happen to read 0^* followed by a 00^* that is R7.

And you happen to read 10^* in the first dimension and 00^* in the second dimension, that is R7. And that is the set of rules, basically, five nodes are replicated in the set pruning trie. So, now, if you convert this into the grid trie. So, how many nodes will be eliminated from the trie structure, basically, as many nodes are replicated in the set prune trie, those many nodes are eliminated. So, basically, five nodes will disappear from this trie structure. And those many number of the links, I need to redirect to the respected nodes in the structure.

(Refer Slide Time: 16:18)

Trie based Packet Classification Algorithms:
Grid Trie

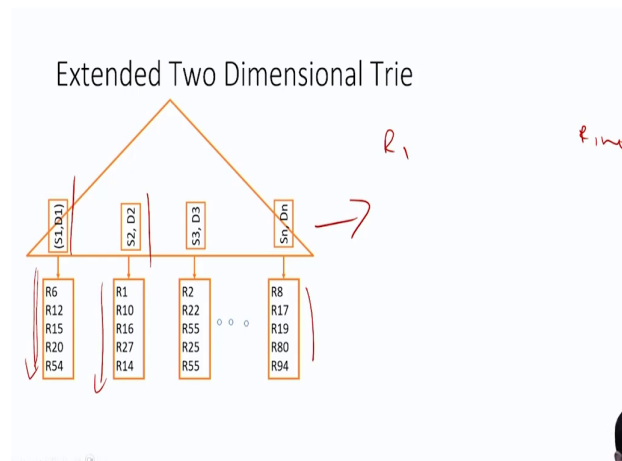
Rule	d1	d2
R1	00^*	110^*
R2	00^*	11^*
R3	10^*	1^*
R4	0^*	01^*
R5	0^*	10^*
R6	0^*	1^*
R7	$*$	00^*

So, here is the one, so the first level and the second level trie structure. So, it is done. Now, I am going to add the links which are missing or replicated in the previous case. So, from this node, remember, only the last level nodes are actually pointing to the node because I can read only one symbol using the traverse in that particular link. So, if there are intermediate nodes, they need to be still replicated in the grid structure, that is what is done here.

Similarly, on the input 1, so you are replicating the node R4, so I added a link to the R4. And from this one on 0 on the left-hand side, it was going to R5 so I added the link to the node R5 itself with the label 0. And from here, on the input 0 is going to the R7 I added that, and from here on the input 0 is going to the R7 that is also replicated. So, that is the whole structure. So, basically there are five links that are added to the trie structure.

So, this is one optimization that you can bring by little modification to the hierarchical trie and the set prune trie. So, that is how this classifier is actually going to work using this grid trie. So, if you have a very large number of nodes being replicated in the trie structure, then grid trie will actually help in the reduction of the number of nodes. So, with that optimization, let us move on to the next type of optimization that you can bring on to the table.

(Refer Slide Time: 18:04)



This optimization is something called the extended two-dimensional trie. What is shown here is a set of rules organized according to something called the source and destination IP address. So, there is a laundry list of the IP Addresses, let us say I got the rules from R1 to R100. What the idea here is when you take the set of the rules and most commonly the source and destination IP address of the two components of any rules that you write.

Even if you have 1000s of rules and when you pick up one particular packet and inspect the source and destination IP address fields, not many numbers of the rules actually match among the combinations that I have. I have 1000 rules in all, but if I pick up one particular source and destination IP address combination, there are only a handful number of rules which actually are candidates for matching.

So, the idea here is I segregate or divide all the rules in my classifier first based on the source and destination IP address combination. What I mean by that is: these are the set of rules from R6, R12, R15, R20, and R54, and the rules which are applicable for this unique source and destination IP address combination. And these are the set of rules which are applicable for source and destination IP address S2 and D2 combination and so forth.

So, if I have n number of the rules in my classifier, I find out first in the first place, what are the unique combination of the source and destination IP address fields or unique prefixes of the source and destination IP address field and then segregate the rules which are applicable for that combination only, you might have some reduction in the case, nevertheless I am saying the S1 D1 it is actually the complete IP address.

The prefix part, you can expand and then bring the unique combination of the source and destination IP address. Then organize the rules which are applicable for that particular combination into the structure. So, then, we know that there are some data structures that allow you to do the two-dimensional comparison, for example, the hierarchical trie or the grid trie or the other structure I can use.

So, assuming I have multiple dimensions, source and destination IP address are the two components, but there are a bunch of other field dimensions in my classifier, and I build a data structure for two-dimensional classification in the first place. So, that is the first level identification that you do. And then, in the second case, what I do is once I narrow down which of these among the list you need to compare, linearly I can go and search one by one because assuming these are not many numbers, I can afford to do that.

So, that is the whole idea behind the extended two-dimensional trie structure. So, basically, you do a first search on the source and destination IP address fields which are commonly found in most of the rules and then you go ahead and check linearly one by one the other fields which are applicable in the set, otherwise this is exactly the same as that of the hierarchical trie and the other grid trie.

So, you can use any of those data structures to build such. This is a heuristic approach. This originated after studying many classifiers formed in the commercial routers of the network service providers. The people found out that this is indeed a better method to organize these so that I can minimize the overall number of such operations. So, that is the heuristic approach. This also you can bring into to increase the speed of the classification that the routers do.

(Refer Slide Time: 22:47)

Trie based Packet Classification Algorithms:
Field Level Trie

{5-11}

Data Structure

1. Every node in the trie has a set of applicable rules
2. A node at level i generate child nodes considering the values in that field
 1. If the field value is in prefix format then the child nodes include all applicable rules for that prefix
 2. If the field value is in range format then the child nodes indicate unique range
3. If two nodes at i th level have children having common applicable rules then only one child node is created



So, with that, let us move on to the next data structure. This is something different, something called the field level trie structure. So, the field level trie structure, basically, it is also kind of the heuristic, but what it does is it starts with all the rule set in the classifier. In the beginning, you assume that every rule is an equal potential candidate.

And I take one dimension at a time and then try to segregate what are the rules applicable if that prefixes match these many reductions; these are the only candidates which need to be further examined. So, that is what the field-level trie structure does. So, first, let us understand what the data structure is and then we will take an example and see how the trie structure looks like.

So, every node in the trie structure has a set of applicable rules. Earlier in the previous case, every path and every link between the nodes actually had one bit of labeling. Instead of that, what you do here is a multiple number of bits, in fact, the whole prefix itself. And then if I, if this prefix is matched, these are the number of the only rules you need to examine.

So, I start with the root node, which is having the entire set of rules in the classifier, and then let us say 00^* is one prefix at the dimension d_1 . With 00^* , what are the now applicable rules? A subset of the rules in the root node would be the second-level node. So, like this, I continue doing that. One dimension, second dimension, third dimension, whatever the number of dimensions I have, I keep dividing the rule set into multiple pieces.

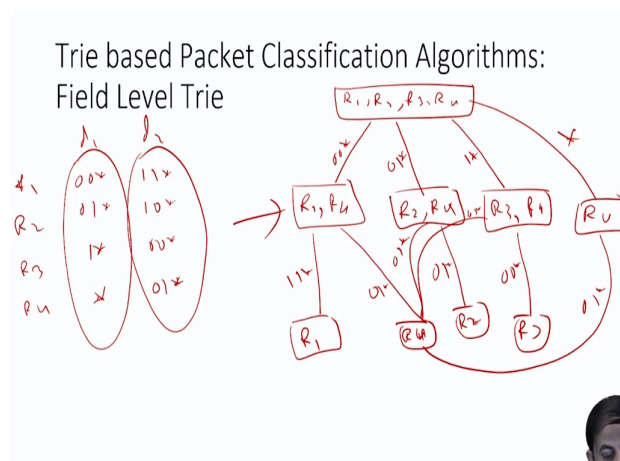
And then finally, when I hit the leaf node, I get one particular rule which is applicable for that particular packet. So, every node in trie structure indicates the applicable rules. Remembering the hierarchical and the set prune trie and the grid trie, at every leaf node when you highlight it, it is only one rule that is applicable, but here the intermediate nodes can indicate more than one number of rules.

So, a node at the level i generate the child nodes considering the values in that particular field. What is the prefix that you are considering right now using that, it is going to add what are the child nodes corresponding to itself. If the dimension $d1$ or $d2$, whatever I am saying, sometimes you will have the source and destination IP address I can express in the form of the prefix. But sometimes you use the other fields like the port numbers, then not necessarily be in the prefix format. If there is a prefix format, then for that prefix, you will have the child node. And if the field value is expressed in the range, particularly if I say that the port number needs to be between 5 and 10. Then for that range, for every unique range, I am going to create the child node.

And at any i th level, if let us say, I take the path one and then say that actually the $R1$ $R2$ are the applicable rules. And then when I go to the second level, only on some prefix, only $R2$ is applicable. And when I traverse the other particular path maybe from that also, I might get on a certain prefix only rule $R2$ is applicable.

So, instead of creating the two nodes at the i th level if there is a rule, which is having the unique combination not necessarily only $R2$ even if you think that $R1$ and $R2$ are the unique combinations at the i th level. Only one node with the label $R1$ and $R2$ is included at that particular level. So, that is how this field-level trie works.

(Refer Slide Time: 27:08)



So, let us see with a simple example how the trie actually looks like. Let us again assume there are two-dimensions d_1 and d_2 , and rule R_1 has the prefix 00^* and 11^* and rule R_2 has 01^* and 10^* and rule R_3 says 1^* and 00^* and rule R_4 says this is $*$ and 01^* . So, what do we need to do as I said, you need to start with the root node, now when you begin, every rule in the classifier is equally likely to be the candidate for the match.

So, now, what are the unique prefixes at the first, I go one dimension at a time, so I take this dimension and look at the unique prefixes. So, for example, here, the first unique prefix is 00^* , I add a link here with 00^* , what are the potential rules that are applicable? So, obviously R_1 is applicable, it depends on the second dimension, but as of now, R_1 is applicable, and also $*$ is a superset of 00^* , so that might also be the candidate, so I write R_1 and R_4 as the potential candidates for this prefix 00^* .

Now, from the four rule node to the two-rule node, we have added, and I keep going further. So, on the prefix 01^* , what are the potential candidate rules applicable? R_2 is one candidate obviously, and so is R_4 . So, because $*$ is a superset of 01^* , so, that is still applicable. And I continue doing this. So, on 1^* , you have got R_3 . And 1^* is not a superset of R_1 and R_2 , no question of that. So, still, 1^* is a subset of the $*$.

So, I include R_4 as well. If none of these prefixes match, then you take the $*$ out and then go to the, now only here R_4 is applicable, so that is the division. So, using the unique prefixes at the first level, or first dimension, you branch out and then create the unique set of rules in the second level. Now, what I am going to do is I am going to examine the second dimension.

And using that as an input to the nodes at this level, I am going to again create further division. So, if this 11* is one applicable prefix for the rule R1, so on the 11*, rule R1 is applicable. So, remember, this R4 is not applicable now because that prefix that requirement is 01*. So, if 01* is found, then the rule R4 is applicable. So, I write that as a separate node. Similarly, on this node combination R2 with leaf R2 R4.

So, if you happen to read 01* in the second level, then R2 is applicable. And on the other hand, if you happen to read 01*, then this rule is applicable. So, this is what I was telling at a particular level. Now, R4 is branching out from the R1 R2 node as well and R2 R4 node as well. We do not repeat R4 many numbers of time, only once, and just add the pointer. So, for R3 as well, on the 00*, only R3 is applicable.

And on the 01* again, I am going to this particular node itself. And again on 01*, so in the second dimension, I am going to this node. So, this is the field level trie structure for this particular classifier with the four rules. So, basically, segregation is shown.

So, this is a case when none of the dimensions or values are expressed in the form of a range particularly to be more precise when you do not have the port value expressed in the form of the ranges, but they are not very unlikely to appear in any of the classifiers.

And from here on the input to 10^* , I am going to node R2, and from here on the input to 00^* I am going to node R3. So, this is the set of nodes that I have for this. So, now for rule R1 what is a port range that is applicable, 4 to 8, I write this range in between 4 to 8, and rule R1 is applicable. And for this case, this exactly needs to be 5. Then rule R4 is applicable.

And this is R2, for R2 this need to be between 1 and 2. On this range, the rule R2 is applicable. For R3, this needs to be exactly 10 and then this is the node. So, this is a simple case where the ranges are not overlapping, but it is not always the case. So, for example, if two similar prefixes 00^* 11^* have the R1, 00^* and 11^* have the R2.

But for the first rule, the port number is 1 to 10, for the second rule port number is 5 to 20, then there is a portion that is actually overlapping. So, some portion is going into the R1, some portion is coming to R2 and some portion is actually overlapping you need to handle those cases as well.