Packet classification is difficult for four fields (Source and Destination IPs, source and destination Ports). Hence, packet classification for two fields namely, source and distimation IPs, is presented with three data structures and algorithms.

An example rule set for the address matching is presented in Table 1. The priority of the rules decreases down the table. That is, even if the longest prefix match is obtained with Rule 5, Rule 1 can supersede since it has the highest priority.

Rule	Destination	Source
R1	0*	10*
R2	0*	01*
R3	0*	1*
R4	00*	1*
R5	00*	11*
R6	10*	1*
R7	*	00*

Table 1

Data Structure 1 :

A destination trie is constructed based on the destination prefixes as shown in the Figure 1. Nodes in the destination trie point to the associated source trie as per the Table 1. For example, 00* is associated with 1* and 11* according to Rules 4 and 5. This association can be seen in the Figure 1.

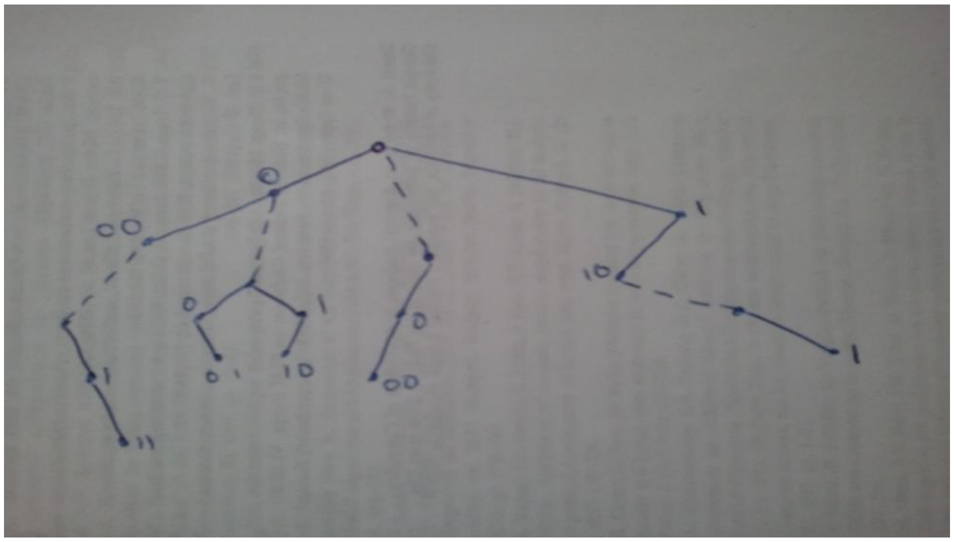

Figure 1.

Algorithm 1 :

1. Given an IP Packet P, a longest prefix match is obtained between P.dest_ip and destination in the constructed trie by walking the trie as deep as possible.
2. Let the node in the trie where the longest prefix match is seen be N.

3. Search for P.source_ip in the source trie pointed to by N by walking down the trie as much as possible.
4. Backtrack

For example, if P.dest_IP = 000….. and source is P.src_ip = 100… . The longest prefix match for the source is obtained at the node 00. Once the destination is matched, the source address(100..) is matched in the source trie pointed by 00. Note that the source_ip can be matched only upto the first bit in the source address. Once it is found that the source_ip cannot be matched any further, the search is performed on the source trie of the parent of N. This is backtracking. Backtracking is done until the root of the destination trie is reached. Backtracking allows the algorithm to find a rule that has higher priority than the rule with the longest prefix.

This algorithm has low storage cost of O(n). But the computation cost is O(n^2) where n is the length of the IP address. The computation cost is more since backtracking searches for the match in the source tries of all the ancestors upto the root of the destination trie.

Data Structure 2 :

For every node in the destination trie that points to a source trie, copy the source tries of all of its ancestors and superimpose on its source trie as shown in the figure 2. The source trie of 00 is merged with the source trie of 0 and the root. Similarly, the source trie of 10 is merged with the source trie of the root.

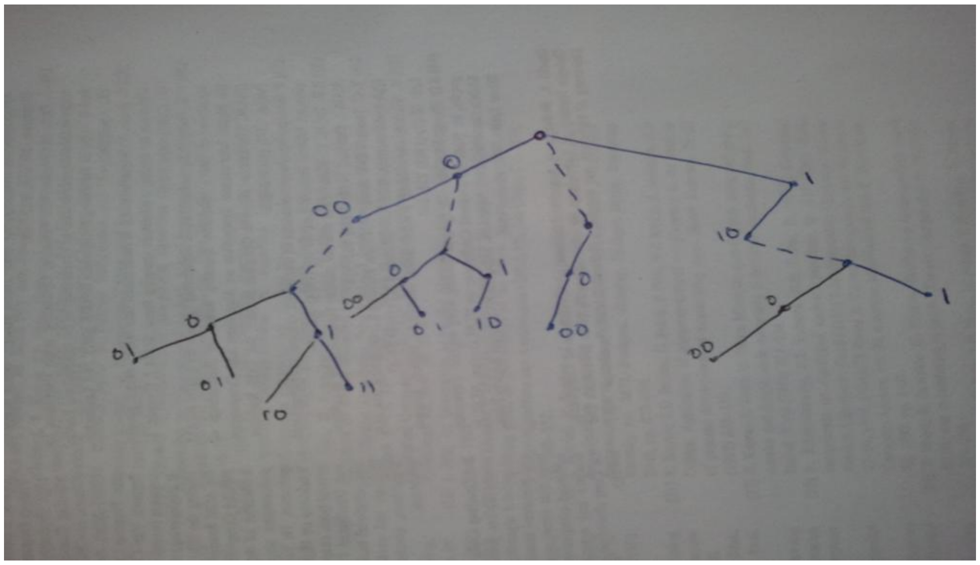

Figure 2.

Algorithm 2 :

The algorithm consists of the steps (1 – 3) of algorithm 1. Note that backtracking is not necessary with this data structure since all the source tries that are needed(source tries of the ancestors) are already merged with the source trie at the current node. For example, inorder to find the match for P.dest = 000.. and P.src = 100 .. , the source trie of 00 has all the required information of the rules to obviate the need to backtrack to the root.

This algorithm has the computational complexity of O(n) since 2n hops are required at the most(n hops for destination match and n hops for source match). But the storage cost can go upto O(n^2).

Data Structure 3 :

A threaded data structure is used to decrease the cost of the storage involved in the data structure 2. The switch pointers are added to the source trie to avoid the copying of the source tries and to avoid backtracking. Switch pointers are added following the algorithm:

1. Start at the leaves of the destination trie.
2. For every source trie pointed to by the leaf of the destination trie, look for the missing node in the nearest ancestor's source trie.
3. Add a switch pointer from the current node pointing to the ancestor's source trie node.

Figure 3 shows the trie with the switch pointers. The '0' node of the source trie of '00' points to the '0' node of the source trie of '0'. Similarly '10' node of source trie of '00' points to the '10' node of source trie of '0'.

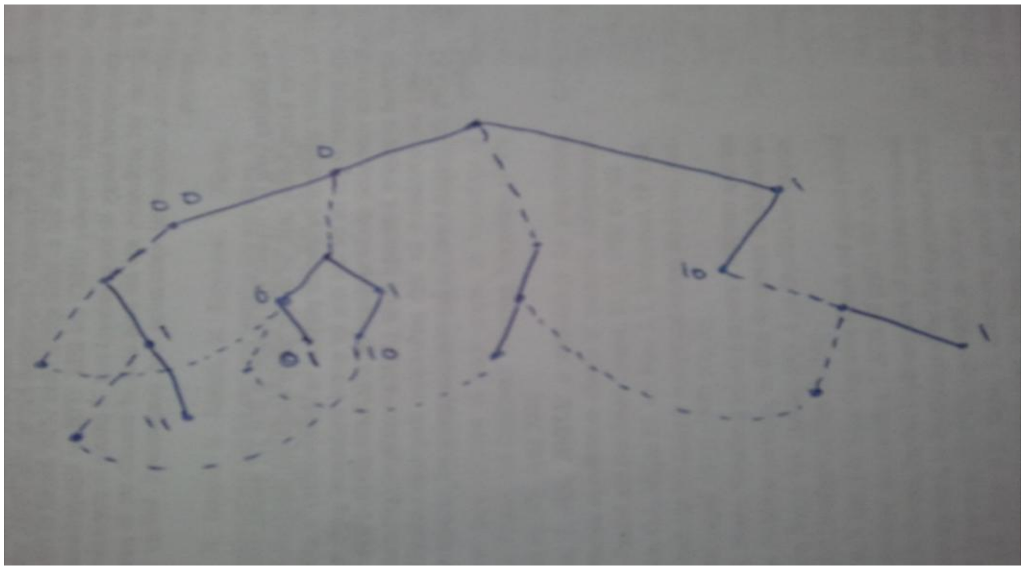

Figure 3.

Algorithm 3 (Best of both worlds) :

1. Follow algorithm 1 from steps 1 – 3.
2. Once a point is reached where the current source trie does not have any node to match, follow the switch pointer to the ancestor's source trie.
3. Continue the same procedure until the root is matched

For example, when searching for destination = 001 and source = 10 in Figure 3, the source trie of '00' is traversed. But since '10' does not exist, the switch pointer is followed to traverse the source trie of '0' where '10' exists.

It can be seen that this algorithm has linear time complexity and linear space complexity.