Efficient Packet Classification using Splay Tree Models

Srinivasan.T, Nivedita.M, Mahadevan.V

Sri Venkateswara College of Engineering, India.

Summary

Packet classification forms the backbone on which a variety of Internet applications operate, providing QoS, security, monitoring, and multimedia capabilities. In order to classify a packet as belonging to a particular flow or set of flows, network nodes perform a search over a pre-defined rule-set, using multiple fields in the packet as the search key. It is being widely used today in high-speed packet forwarding in the Internet by the routers. To improve the performance of the traditional routers requires faster and more efficient lookup techniques for packet classification and switching. In this paper, we propose an efficient, high-speed and scalable packet classification technique using splay tree models and employing prefix conversion methods, called Splay Tree based Packet Classification (ST-PC). It is aimed at overcoming some of the performance limitations of the previously known techniques. Theoretical analysis of the proposed technique illustrates the high performance gains in terms of space and time complexities, over the well-known techniques in literature and their corresponding data structures.

Key words:
Packet Classification, Routers, Classifier, Prefix Conversion, Splay Trees.

1. Introduction

The process of classifying packets into "*flows*" in routers, firewalls, packet filters etc., is called packet classification. Packet classification is used in a variety of applications such as security, monitoring, multimedia applications etc. These applications operate on packet flows or set of flows. Therefore these nodes must classify packets traversing through it in order to assign a flow identifier, called as Flow ID [1], [7], [12].

Packet Classification starts by building a classifier of rules or filter table, then searching that table for a particular filter or a set of filters that match the incoming packets. Each filter consists of a number of filed values. The field values may be an address filed such as source, destination addresses or a port field namely source, destination ports or protocol type [1].

The main research issues in the design of optimal packet classification techniques are: to increase the packet classification speed, to increase the update performance speeds for new rules, to decrease the storage requirements for caching these rules.

The rest of the paper is organized as follows. Section 2 discusses the different techniques known in literature. Section 3 presents our proposed technique in detail. Section 4 illustrates the theoretical analysis of the proposed technique with respect to other well known techniques, while Section 5 concludes.

2. Related Work

Numerous algorithms and architectures for packet classification have been proposed [1], [2], [3], [4], [6], [7], [14]. A taxonomy that breaks the design space into four regions based on the high level approach to the problem is also discussed in the above said papers.

2.1. Hierarchical Tries

A d-dimensional hierarchical radix trie is constructed recursively as follows [1], [7]. If 'd' is greater than 1, we construct a 1-dimensional trie, called the F1-trie. This trie is constructed on the set of prefixes $\{R_{j1}\}$, belonging to dimension F1 of all rules in the classifier $C = \{R_j\}$. For each prefix in the F1-trie, a (d-1)-dimensional hierarchical trie, T_p, is recursively constructed, on those rules which specify exactly 'p' in dimension F1, i.e., on the set of rules $\{R_j: R_{j1} = p\}$. Prefix 'p' is linked to the trie T_p using a *next-trie* pointer. The Table 1 below shows an example classifier with the destination & source addresses as fields. The hierarchical trie data structure for the classifier in Table 1 is given in Figure 1.

Table 1: Classifier

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

Classification of an incoming packet ($v_1, v_2, \ldots, v_d$) proceeds as follows. The query algorithm first traverses the F1-trie based on the bits in v_1. At each F1-trie node encountered, the algorithm follows the next-trie pointer (if present) and traverses the (d-1) - dimensional trie.

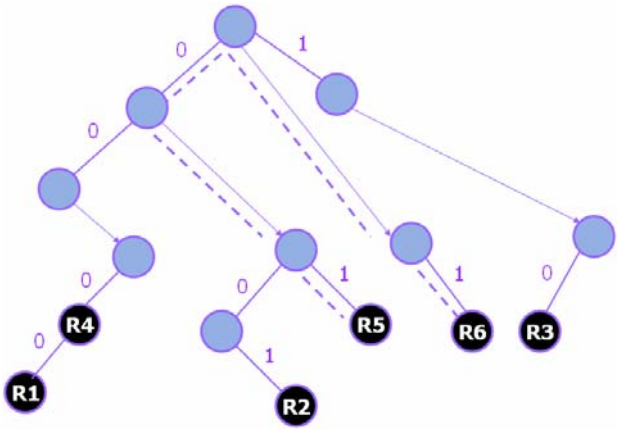

Fig. 1 Hierarchical Tries.

2.2 Splay Trees

Splay trees are self – balancing (or) self – adjusting binary search trees [2]. It has special update and access rules. Every time we access a node of the tree, whether for retrieval or insertion or deletion, we perform radical surgery on the tree, resulting in the newly accessed node becoming the root of the modified tree. This surgery will ensure that nodes that are frequently accessed will never drift too far away from the root whereas inactive nodes will get pushed away farther from the root.

When we access a node, we apply either a single rotation or a series of rotations to move the node to the root. The biggest advantage of using Splay trees is that it does not require height or balance factors as in AVL trees and colors as in Red-Black trees. Informally, one can think of the splay trees as implementing a sort of LRU policy on tree accesses i.e. the most recently accessed elements are pulled closer to the root; and indeed, one can show that the tree structure adapts dynamically to the elements accessed, so that the least frequently used elements will be those furthest from the root. But remarkably, although no explicit balance conditions are imposed on the tree, each of these operations can be shown to use time *O(log n)* on an *n*-element tree, in an amortized sense [2].

There are six rotations possible in a splay tree. They are:

1. Zig Rotation
2. Zag Rotation
3. Zig-Zig Rotation
4. Zag-Zag Rotation
5. Zig-Zag Rotation
6. Zag-Zig Rotation

Figure 2 depicts the cases under which these rotations will be applied. In this Figure, the node ‘x’ is the node with respect to which the splaying is done. Node ‘p’ is the parent node of node ‘x’. Node ‘g’ is the grandparent node of node ‘x’, which is the parent node of node ‘p’. Figure 3 shows an example of splaying which employs a Zig rotation.

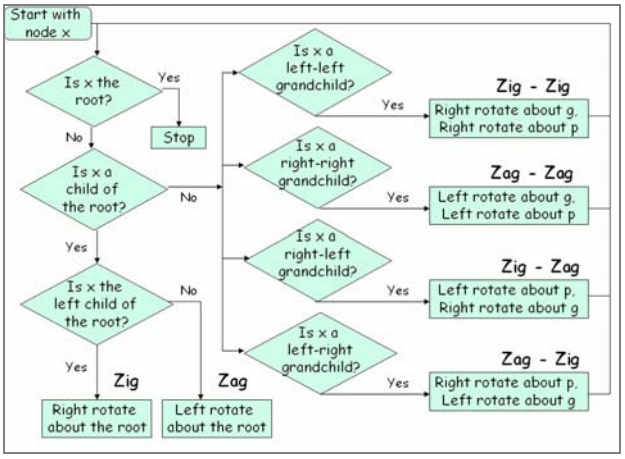

Fig. 2 Rotations in a splay tree.

Consider an example of splaying as shown in Figure 3. Here, when we access node 16, it has to become the root of the tree by the property of a splay tree. The tree will reorder it self such that node 16 becomes the root of the tree, node 0 becomes its left child and node 63 becomes its right child. Also, node 32 will no longer be the right child of node 16, but becomes the left child of node 63.

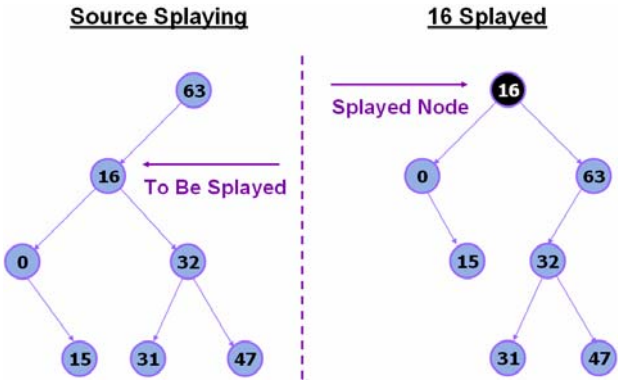

Fig. 3 Example of splaying.

3. ST-PC Technique

In this paper, we propose a novel representation of the input set and build a tree from this input set. The basic idea is to convert the set of prefixes into integers. Firstly, we find out the lower and upper bounds for each prefix in the source address. Then we convert these values to integers and store them in a database. The same procedure is carried out for each of the prefixes of the destination and stored in the database. While classifying incoming packets, we reject packets if they do not match the constraints. Finally we find out the best matching filter (rule) among the various filters for the valid packets.

3.1. Design Methodology

The proposed work is a basic extension of the Hierarchical Tries. Here, we convert each of the source and destination prefixes into integer ranges. Then the corresponding tree is constructed. The greatest advantage of this approach is that the prefix specification can be extended to any number of bits.

Table 2: Source Prefix Conversion

Filter	*Source Prefix*	*Lower Bound*	*Upper Bound*	*Start*	*Finish*
F1	01*	010000	011111	16	31
F2	1*	100000	111111	32	63
F3	10*	100000	101111	32	47
F4	01*	010000	011111	16	31
F5	00*	000000	001111	0	15
F6	*	000000	111111	0	63

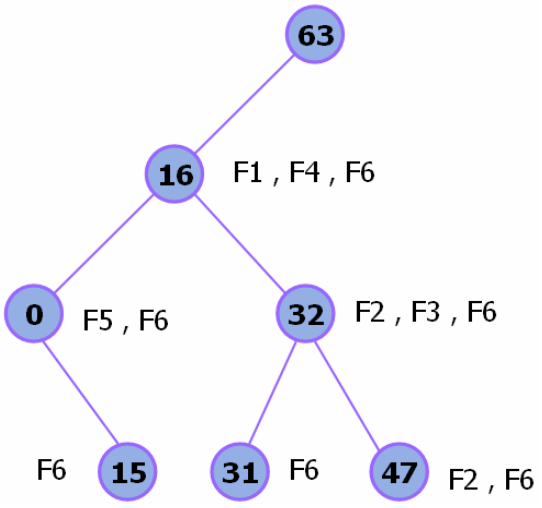

Fig. 4 Source splay tree.

We first compute the lower and upper bounds for each of the prefixes in the source address as shown in Table 2. Then a source splay tree is constructed with the bounds converted to integers, which is shown in Figure 4. Similarly, using the destination addresses, a destination splay tree is constructed as shown in Table 3 and Figure 5.

Now the source and destination splay trees are to be linked. For this, we connect each leaf of the source splay tree to the root of the destination splay tree. This connection pointer is similar to a *Next-Trie* pointer used in a hierarchical trie data structure [1], [7], [12].

Table 3: Destination Prefix Conversion

Filter	*Source Prefix*	*Lower Bound*	*Upper Bound*	*Start*	*Finish*
F1	01*	010000	011111	16	31
F2	01*	010000	011111	16	31
F3	0*	000000	011111	0	31
F4	00*	000000	001111	0	15
F5	1*	100000	111111	32	63
F6	1*	100000	111111	32	63

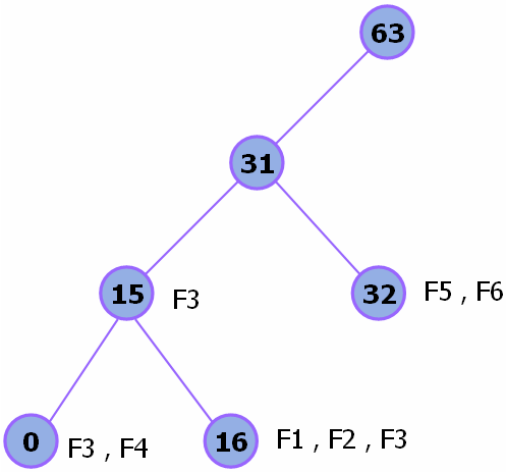

Fig. 5 Destination splay tree.

In order to find out the best matching filter, we convert the source and destination prefixes of the search packet to integer values and then begin searching. We first see the integer source value of the packet and find out the filters whose lower bound is less than the packet's source integer value and whose upper bound is greater than the packet's destination integer value. In other words, we pick out all those filters within whose range the search packet's value lies. Similarly, we find out the matching filters for the destination's integer value of the search packet. Then, we perform a simple comparison between the filters that have matched the source and destination of the packet separately.

To construct a source search table and a destination search table, we get the set of distinct integer values in both the source and destination trees and arrange them in the ascending order. We now find the set of filters that match all points between the first and second integer values, between the second and third value and so on until the entire table is constructed as shown in Tables 4 and 5.

An example search packet is also shown below these tables. We firstly convert the prefixes of the search packet

to integers and then find the separate source and destination matching filters and finally find the final set of matching filters for that particular search packet.

Search

Table 4: Source Search Table

Src Point.	Filter > than point
0	F5 , F6
15	F6
16	F1 , F4 , F6
31	F6
32	F2 , F3 , F6
47	F2 , F6

Table 5: Destination Search Table

Dest Point.	Filter > than point
0	F3 , F4
15	F3
16	F1 , F2 , F3
31	--
32	F5 , F6

Example Search

(000110, 101100) => (06 , 44)
=> { (F5 , F6) ; (F5 , F6) }
=> { F5 , F6 }

3.2 ST-PC Evaluation Architecture

Figure 6 shows the various stages through which our proposed prototype will go by. We shall describe each prototype briefly now.

Prefix Conversion: All the existing algorithms so far have been constructing the trie for the input filter set using prefixes of the form 00*, 01*, 1* and so on. That is, they used a basic binary trie. A trie is a data structure where there can be only 2 inputs, namely a ‘ 0 ’ or a ‘ 1 ’. All the ‘ 0 ’ transitions from a node are to the left of the node whereas all the ‘ 1 ’ transitions from a node are to the right of the node. The major disadvantage of this method is that the amount of space required for the trie is tremendous. Moreover, it grows exponentially as the number of bits in the prefix increases. Our new method just stores them as ranges (integers).

Rule Storage: The input filter set must be formed in conformance to the earlier decided parameters. After this is done with, we need to store this input filter set into the router’s database. Now, we are ready to move onto the next step in our architecture.

Packet Conversion: We check whether the incoming filter’s parameters match the ones which are present in the router’s database. If they are as per the specifications, then we go on and check whether the parameters are valid.

Matching Filter(s) Search: The main objective at this stage is to select the set of matching filter(s) for the incoming packet. The **best matching filter** can be found out by performing a simple search through the database. Based on the matching filter, we route the packet through that particular route / routes (line connecting the source to the destination).

There are several constraints on the values of the parameters such as follows:

i. The prefixes entered must be in the binary form, i.e in the form of strings of 0’s and 1’s for certain algorithms.
ii. The dimensions must be of the specified number of bits, i.e as per what we had decided previously.

If the incoming packet matches and satisfies all these conditions, then the packet is passed on, i.e we carry on to the next stage of our architecture, else we reject the packet with the appropriate error message.

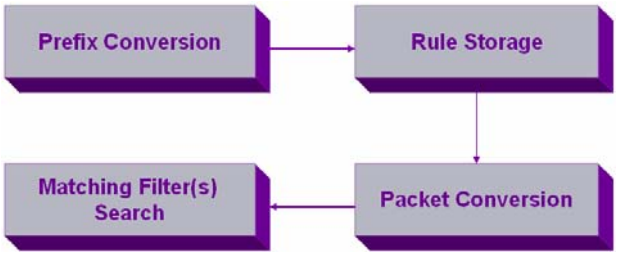

Fig. 6 ST-PC evaluation architecture.

4. Complexity Analysis

We analyze the proposed packet classification technique here, while also comparing space-time requirements with other well known techniques.

For our entire complexity analysis, we use a few notations as follows:
r = Number of rules,
n = Number of nodes in the tree,
k = Number of bits to represent the prefix,
t = Number of times a prefix is repeated,
β = Depth of a certain node in the tree,
m = Number of prefix lookups.

4.1 Space complexity

4.1.1 Best Case

For the best case analysis, for any source or destination prefix, assume that none of the 'k' bits are fixed. That is, all the bits can be arbitrarily either 0 or 1. In other words, the prefix is denoted by a (*). So, the best case number of nodes in a binary trie is equivalent to 1. For any source or destination prefix, assume that all the 'k' bits are fixed. That is, all the bits are either 0 or 1. In other words, the prefix will have a length of 6. Trivially, number of unique rules is 1. The Best Case number of nodes in a Splay Tree is equivalent to 1. **Nodes in Binary Trie = Nodes in Splay Tree.**

4.1.2 Worst Case

In the worst case, for any source or destination prefix, assume that all the prefixes are distinct. In this case, the number of nodes will be $2^{k+1} - 1$. Assume that all 'k' bits of each of the 'r' rules are distinct. The number of nodes in this case will be (k * r) + 1. So, the actual number of nodes in a binary trie in the worst case will be a minimum of these 2 values i.e. MINIMUM ($2k^{+1} - 1$, (k * r) + 1).

For any source or destination prefix, assume that all the prefixes, and hence the bounds (integers) are distinct. In this case, the entire range of values is possible for various combinations of prefixes, and hence the number of nodes will be 2^k. Next, assume that only a few prefixes occur in the rules. Since we convert these prefixes to integer values, the number of distinct values will be at most 2*r. Hence the number of nodes in this case will be 2*r. So, the actual number of nodes in a splay tree in the worst case will be a minimum of these 2 values i.e. MINIMUM (2k, 2 * r). **Nodes in Splay Tree < Nodes in Binary Trie.**

4.1.3 Graphical representation

Case 1:

Figure 7 compares the number of bits in the prefix (k) with the number of nodes (n) required for both the Binary trie and Splay tree. Here, the number of rules remains a constant which we assume to be 6.

As it can be seen from the graph, the number of nodes is the same for a value of k = 0. For values greater than 0 and less than 4, there is a small variation in the number of nodes. But beyond k = 4, 'n' increases linearly for a Binary Trie; whereas it remains at a constant value of n = 12. Thus, this clearly shows that the number of nodes required significantly reduces for a Splay tree when number of rules is a constant.

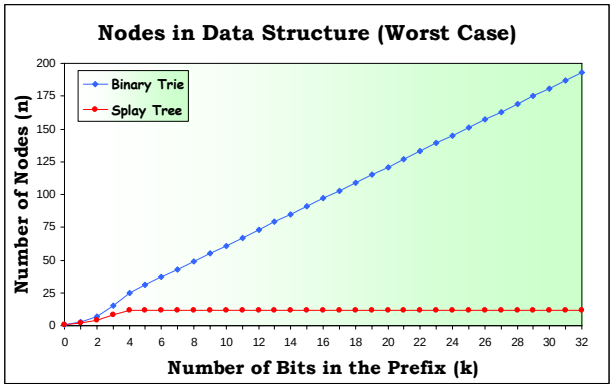

Fig. 7 Number of bits vs. Number of nodes

Case 2:

Comparing, the number of rules (r) with the number of nodes (n) as in Figure 8, the number of nodes is the same for a value of k = 0. For values greater than 0 and less than 21, the number of nodes required increases linearly, after which it remains at a constant value of 127 for a Binary trie. But the number of nodes required for a Splay tree is much lower for all values when compared to the Binary trie.

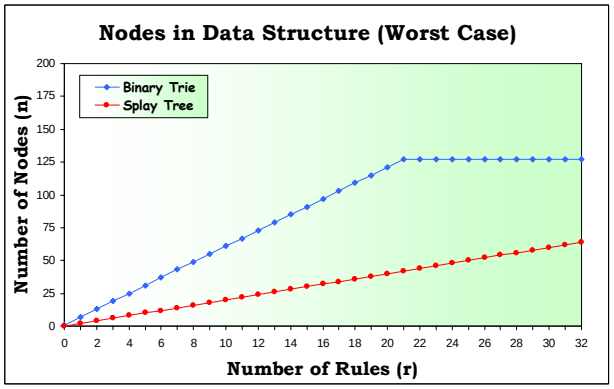

Fig. 8 Number of rules vs. Number of nodes

4.2 Time complexity

Case 1:

Assume that there are 'm' unique accesses, i.e. all the prefix searches are unique. The search time of a node for a binary trie is log (2n) and the search time of a node for a splay tree is log (n). This is true as the number of nodes in the binary trie is twice that of a normal binary search tree, as the nodes of interest occur only at the leaves of the trie. For 'm' accesses, the search times for a binary trie and a splay tree will be m * log (2n) and m * log (n) respectively. **Time of Splay Tree < Time of Binary Trie.**

Case 2:

Assume that there are (m – t) unique accesses. That is, (m – t) searches are unique. Hence, search time for a binary trie in this case is m log (2n) as before. Consider the case of a Splay Tree. Here, (m – t) unique operations will take a time of (m – t) log (n). For the remaining 't' accesses, the time required will be t * log β, where log β <= log n. Hence, the total time required for the splay tree is [((m – t) log n) + (t * log β)]. **Time of Splay Tree < Time of Binary Trie.**

4.2.1 Frequency distribution (t)

This is governed by the number of times the prefixes are repeated. Consider the following example.

Let (a , b , c , d , e) be a sequence of 5 accesses of prefixes.

Let (a , a , a , b , b) be another sequence of 5 accesses of prefixes.

For the first case, t = 0 since no prefixes are repeated. But for the second case, t = 2 + 1 = 3 since 'a' is repeated twice and 'b' is repeated once. Thus, the time complexity for Splay Trees reduces if there are repetitions in prefixes.

4.2.2 Temporal distribution (β)

This is governed by the compactness of the occurrence of the prefixes. Consider the following example.

Let (a , b , a , b , a , b) be one sequence of accesses in which prefixes occur.

Let (a , a , a , b , b , b) be another sequence of accesses in which prefixes occur.

In the first case, the value of β is greater than 1, for all accesses; since for each access the nodes may be at random depths within the tree. But, in the second case, the value of β is 1 for all repetitive accesses; since the nodes move closer to the root after each access due to splaying.

4.2.3 Graphical representation

Figure 9 compares the number of nodes (n) with the corresponding access time (t) required for both the Binary trie and Splay tree for a unique sequence of accesses when the number of prefix lookups remains a constant which we assume to be 25.

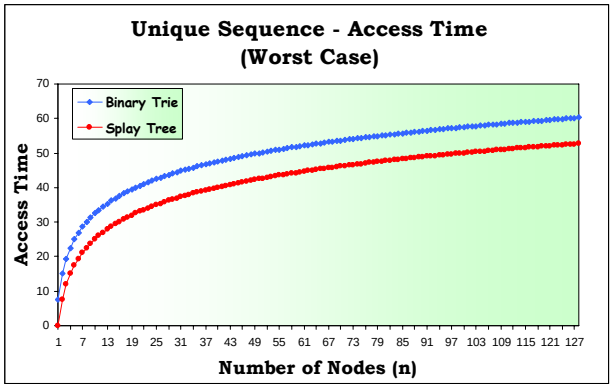

Fig. 9 Access time in a unique sequence

Figure 10 compares the number of nodes (n) with the access time (t) required for both the Binary trie and Splay tree for a repeating sequence of accesses when the number of times a prefix is repeated remains at a constant value of t = 10 and depth of a node in the tree also remains at a constant value of β = 5. The number of prefix lookups also remains a constant value of m = 25. Here again, the performance of splay trees is better than that of binary tries.

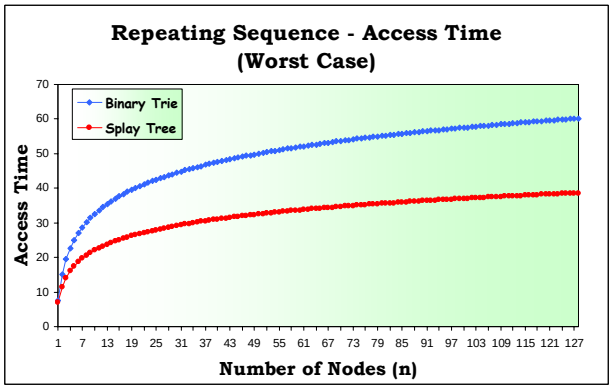

Fig. 10 Access time in a repeating sequence

4.2.4 Comparison

Table 6 gives a brief overview of the complexities of both the binary tries and splay trees. As it can be seen from the table, the space complexity is the same for both the data structures in the best case. In the worst case, the complexity of the splay trees outperforms that of the binary tries. Similarly, the time complexity of the splay trees is much less than that of the binary tries in both the best case and the worst case.

Table 6: Complexity Comparison

Complexity		Best Case		Worst Case	
		Binary Trie	Splay Tree	Binary Trie	Splay Tree
Space	Nodes	1	1	Min [$(2^{k+1} - 1)$, k*r + 1]	Min [2^k , 2*r]
	Edges	0	0	Min [$(2^{k+1} - 2)$, k*r]	Min [$(2^k - 1)$, (2*r) – 1]
Time		m * log(2n)	m * log(n)	m * log(2n)	m * log(n)

5. Concluding Remarks

Fast and efficient packet classification techniques are essential for the design of high-speed routers, and various other applications. In this paper, we propose a novel packet classification technique called Splay Tree based Packet Classification (ST-PC). We use simple primitives in our design, namely splay tree models and prefix conversion methods. The theoretical analysis of the space-time complexity of the proposed work shows significant performance gains over the well known techniques in literature. Future work would focus on optimizing the tree reordering time, inherent to the splaying operation of the proposed data structure.

References

[1] Srinivasan.T, Prasad.S, Prakash.B, "*Dynamic Packet Classification Algorithm using Multi - Level Trie*", Proc of IJIT International Conference on *IT- IT 2004*, Turkey, Dec 2004.

[2] Srinivasan.T, Nivedita.M, Azeezunnisa.A.A, "*Scalable and Parallel Aggregated Bit Vector Packet Classification using Prefix Computation Model*", Conference on Business and Internet, Honolulu, Hawaii, Mar 2006.

[3] Gupta.P, Mckeown.N, "*Packet Classification on multiple fields*", *ACM Sigcomm,* Aug 1999.

[4] Sartaj Sahni, Wencheng Lu, "*Packet Classification Using Two-Dimensional Multibit Tries*", ISCC: 849-854, 2005.

[5] Varghese.G, Srinivasan.V, Suri.S, "*Packet Classification using tuple space search*", *SIGCOMM*, Pages 135-146, 1999.

[6] David E.Taylor, Jonathan S.Turner, "*Scalable Packet Classification using distributed crossproducting of field labels*", WUCSE-38, Washington University, Saint Louis.

[7] Rajaraman.V, Murthy.C.R, "*Parallel Computers – Architecture and Programming*", Prentice-Hall of India, New Delhi, 2003.

[8] Taylor.D, Spitznagel.E, Turner.J, "*Packet Classification using extended tcams*", *IEEE Conference on Network Protocols (ICNP),* 2003.

[9] Varghese.G, Singh.S, Baboescu.F, "*Packet classification for core routers*", Is there an alternative to cams? *IEEE Infocom*, 2003.

[10] Stidialis.D, Lakshmanan.T, "*High speed policy – based packet forwarding using efficient multi - dimensional range matching*", *ACM Sigcomm,* Sep 1999.

[11] Tsuchiya.P, "*A Search Algorithm for table entries with non-contiguous wildcarding*", Bellcore.

[12] Varghese.G, Srinivasan.V, Suri.S, Waldvogel.M, "*Fast Scalable Level Four switching*", Proc. of Sigcomm, 1998.

[13] Turner.J, Plattner.B, Varghese.G, Waldvogel.M "*Scalable High Speed IP Routing Lookups*", Proc. of Sigcomm, 1997.

[14] Baboescu.F, Varghese.G, "*Scalable packet classification*", *ACM Sigcomm,* Aug 1997.

[15] Podaima.J, Gibson.G, Shafai.F, "*Content addressable memory storage device*", United States patent 6,044,005. Sibercore Technologies, Inc, Aug 1999.

Srinivasan T is an Assistant Professor in the department of Computer Science and Engineering at Sri Venkateswara College of Engineering. He holds a Master of Engineering degree in Computer Science. He is a member of the ISTE chapter, India and a member of the Internet Society, USA. His current research interests relate to Parallel and Distributed Systems, Network Protocols and Security, Mobile and Sensor Networks.

Nivedita M received the B.E. degree in Computer Science and Engineering from Sri Venkateswara College of Engineering, Chennai, India. She is a member of the IEEE society. Her research topics include Computer Networks, Mobile Networks, Network Security and IPv6. She is now with Cognizant Technology Solutions, India.

Mahadevan V received the B.E. degree in Computer Science and Engineering from Sri Venkateswara College of Engineering, Chennai, India. His research topics include Wireless Ad-Hoc Networks, Sensory Networks, Computer Networks and Network Security. He is now with Cognizant Technology Solutions, India.