Journal of Information Assurance and Security 4 (2009) 133-141

Dynamic Scheme for Packet Classification Using Splay Trees

Nizar Ben Neji and Adel Bouhoula

Higher School of Communications of Tunis (Sup'Com), University of 7th November at Carthage,
City of Communications Technologies, 2083, Ariana, Tunisia
nizar.bennaji@certification.tn, adel.bouhoula@supcom.rnu.tn

Abstract: Many researches are about optimizing schemes for packet classification and matching filters to increase the performance of many network devices such as firewalls and QoS routers. Most of the proposed algorithms do not process dynamically the packets and give no specific interest in the skewness of the traffic. In this paper, we conceive a set of self-adjusting tree filters by combining the scheme of binary search on prefix length with the splay tree model. Hence, we have at most 2 hash accesses per filter for consecutive values. Our proposed filter is adapted to easily assure exact matching for protocol field, prefix matching for IP addresses, and range matching for port numbers. Also, we use the splaying technique to optimize the early rejection of unwanted flows, which is important for many filtering devices such as firewalls.

Keywords: Packet Classification, Binary Search on Prefix Length, Splay Tree, Early Rejection**.**

1. Introduction

In the packet classification problems we wish to classify the incoming packets into classes based on predefined rules. Classes are defined by rules composed of multiple header fields, mainly source and destination IP addresses, source and destination port numbers, and a protocol type.
In order to deal with the huge traffic, it is necessary to have firewalls and QoS routers with fast link speed, high packet forwarding rate and especially high classification performance with minimum cost.
On one hand, packet classifiers must be constantly optimized to cope with the network traffic demands. On the other hand, few of the proposed algorithms process dynamically the packets and the lack of dynamic packet filtering solutions has been the motivation for this research.
Our work is in the area of optimizing the performance of the packet classifiers by taking into account the relative frequency of the incoming values. Our study shows that the use of a dynamic data structure is the best solution to take into consideration the skewness in the traffic distribution. In order to achieve this goal, we adapt the splay tree data structure to the binary search on prefix length algorithm. Hence, we have conceived a set of dynamic filters for each packet header-field to minimize the average matching time. The proposed algorithm is able to achieve good performance in a practical environment and it can significantly improve the worst-case performance. Many recent works on IP lookup and packet classification problems use the basic scheme of binary search on prefix length proposed by Waldvogel et al. [1] such as the packet classification algorithm proposed by Hyesook Lim et al. [2] which applies binary search on prefix length to the area-based quad-trie and the multidimensional packet classification scheme proposed in [3]. Thus, optimizing the basic scheme of Waldvogel is important because the latter has now been widely used in many packet classification schemes. In addition, many of the data structures used for the representation of filtering rules are static and do not process dynamically the packets.
On the other hand, discarded packets represent an important part of the traffic treated then reject by a firewall. So, those packets might cause more harm than others as they traverse a long matching path before they are finally rejected by the default-deny rule. Therefore, we use the technique of splaying to reject the maximum number of the unwanted packets as early as possible.
The rest of the paper is organized as follows. Section 2 describes the previously published related works and gives a brief description of the splay trees. Section 3 explains the proposed scheme and our optimizing techniques. In section 4 we illustrate the complexity analysis and the experimental results of our proposed algorithm. At the end, in Section 5 we present the conclusion and our plans for future work.

2. Previous Work

Since our proposed work in this paper applies binary search on prefix length with splay trees, we describe in this section the basic scheme of binary search on prefix length algorithm in detail. After that, we present the main properties of the splay tree data structure then we give an example of a previous dynamic scheme based on this data structure called Splay Tree based Packet Classification (ST-PC) [4]. Finally, we present a proposed early rejection technique used to maximize the rejection of the unwanted packets.

Rule no.	**Src Prefix**	**Dst Prefix**	**Src Port**	**Dst Port**	**Proto.**
R1	01001*	000111*	*	80	TCP
R2	01001*	00001*	*	80	TCP
R3	010*	000*	*	443	TCP
R4	0001*	0011*	*	443	TCP
R5	1011*	11010*	*	80	UDP
R6	1011*	110000*	*	80	UDP
R7	1010*	110*	*	443	UDP
R8	110*	1010*	*	443	UDP
R9	*	*	*	*	*

Table 1. Example of a Rule Set with 8-bit prefixes for the source and the destination IP addresses fields.

Received June 10, 2009

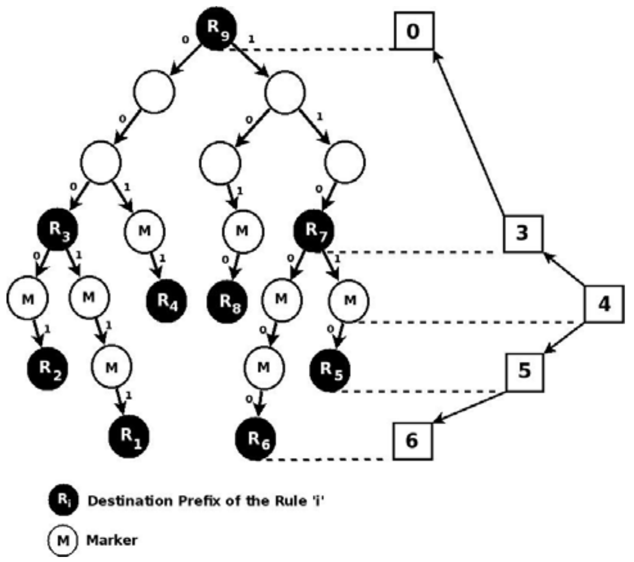


Figure 1. The scheme of binary search on prefix length applied to the destination address field of Table. 1

2.1 Binary Search on Prefix Length

Waldvogel et al. [1] have proposed the IP lookup scheme based on the binary search on prefix length technique. Their scheme performs a binary search on hash tables organized by prefix length. The use of the binary search gives a logarithmic number of searches and their scheme scales well as the filtering list size increases.

Each hash table in their scheme contains prefixes of the same length together with markers for longer-length prefixes (Figure 1). In that case, IP Lookup can be done with O(log(L_{dis})) hash-table searches, where L_{dis} is the number of distinct prefix lengths and L_{dis} <W-1 where W is the maximum possible length, in bits, of a prefix in the filter table. Note that W=32 for IPv4, W=128 for IPv6. Figure 1 shows a binary tree for the destination prefix field of Table 1 and the access order performing the binary search on prefix lengths proposed in [1].

Many other works were proposed to perform this scheme. Srinivasan and Varghese [5] and Kim and Sahni [6] have proposed ways to improve the performance of the binary search on lengths scheme by using prefix expansion to reduce the value of L_{dis}, and the complexity of the controlled prefix expansion algorithm proposed in [5] is $O(NW^2)$, where N is the number of prefixes. Whereas the algorithm of [6] minimizes storage requirements but takes $O(NW^3 + kW^4)$ time, where k is the desired number of distinct lengths. Broder and Mitzenmacher [7] proposed an algorithm using multiple hash functions (two hash functions) to improve the lookup performance of the Waldvogel’s scheme.

In addition, an asymmetric binary search was proposed to reduce the average number of hash computations [1]. This tree basically inserts values of higher occurrence probability (matching frequency) at higher tree levels than the values of less probability. In fact, we have to rebuild periodically the search tree based on the traffic characteristics.

Also, a rope search algorithm was proposed in [1] to reduce the average number of hash computations but it increases the rebuild time of the search tree because it uses pre-computation techniques to fulfill this goal. So we have O(NW) time complexity when we rebuild the tree after an entry insertion or deletion in the list of rules.

The basic algorithm of binary search on prefix length is widely used to improve the performance of several packet classification schemes implemented in high speed packet classifiers such as the area-based quad-tree algorithm which consists of a two dimensional tree using source and destination prefix fields and applies binary search on levels [2]. It constructs multiple disjoint tries depending on relative levels in rule hierarchy to avoid the pre-computation required in the binary search.

Also, a multidimensional classifier similar to the one-dimensional scheme of Waldvogel was recently presented [3] and it also consists of binary search on a collection of hash-tables.

2.2 Dynamic Packet Classification Schemes

2.2.1 Splay Tree Based Scheme

The Splay Tree is a data structure invented by Sleator and Tarjan [8]. It is an ordered binary tree: for every node x, every element in the left sub-tree of x is ≤ x, and every node in the right sub-tree of x is ≥ x.

On a n-node splay tree, all the standard search tree operations have an amortized time bound of O(log(n)) per operation, where by “amortized time” is meant the time per operation averaged over a worst-case sequence of operations. When we access a node, we apply either a single rotation or a series of rotations to move the node to the root.

The most interesting aspect of this structure is that, unlike balanced tree schemes such as 2-3 trees or AVL trees, it is not necessary to rebalance the tree explicitly after every operation it happens automatically with the use of splaying rotations.

In the literature, there are other schemes implemented in the high-speed packet classifiers based on the splay tree data structure such as the Splay Tree Packet Classification Technique (ST-PC) [4].

The idea of the ST-PC is to convert the set of the prefixes into integer ranges as shown in Table 2 then we put all the lower and upper values of the ranges into a splay tree data structure. In the other hand, we have to store in each node of the data structure all the matching rules as shown in Figure. 2. The same procedure can be repeated for the other packet's header fields the resulting splay trees are to be linked to each other to determine the corresponding action to take for the incoming packets.

Rule no.	Dst Prefix	Lower Bound	Upper Bound
R1	000111*	28	31
R2	00001*	8	15
R3	000*	0	31
R4	0011*	48	63
R5	11010*	208	215
R6	110000*	192	195
R7	110*	192	223
R8	1010*	160	175
R9	*	0	255

Table 2. Conversion into ranges of the 8-bit destination prefixes.

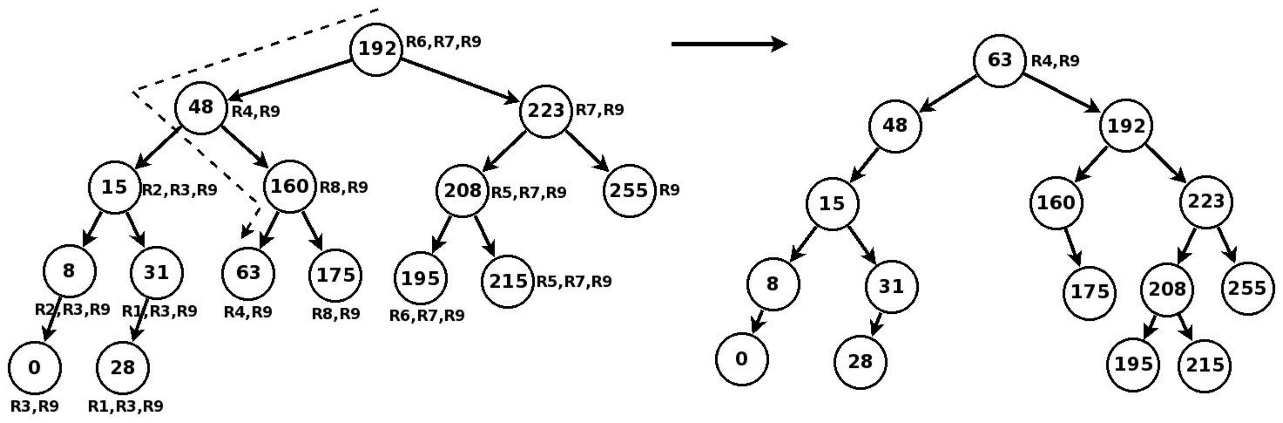


Figure 2. Splaying the newly accessed node in the destination splay tree constructed as shown in Table 2.

2.2.2 Other Dynamic Schemes

Gupta et al. [9] proposed an efficient data structure to exploit the bias in access pattern: they use the extended binary search tree to exploit the difference in the probabilities with which the various leaves of the tree (where the intervals are stored) are accessed by incoming packets in order to speedup the lookup process. Their technique achieves faster lookups for more frequently accessed keys while bounding the worst case lookup time, in fact it is near optimal under constraints on worst case performance. However, this scheme needs to be rebuilt periodically to reflect the changes in access patterns, which can be inefficient for bursty environments.

Funda Ergun et al. [10] introduce a new dynamic data structure to exploit biases in the access pattern which tend to change dynamically. Their data structure, called the biased skip list (BSL), has a self-update mechanism which reflects the changes in the access patterns efficiently and immediately, without any need for rebuilding.

Sahni and Kim [11], [12] develop data structures, called a collection of red-black trees (CRBT) and alternative collection of red-black trees (ACRBT), that support the three operations (Lookup, Insert and Delete) of a dynamic longest-matching prefix-tables (LMPT) in O(log n) time each. The number of cache misses is also O(log n).

In [12], Sahni and Kim show that their ACRBT structure is easily modified to extend the biased-skip-list structure of Ergun et al. [10] so as to obtain a biased-skip-list structure for dynamic LMPTs. Using this modified biased skip-list structure, lookup, insert, and delete can each be done in O(log n) expected time and O(log n) expected cache misses. Like the original biased-skip list structure of [10], the modified structure of [12] adapts so as to perform lookups faster for bursty access patterns than for non-bursty patterns. The ACRBT structure may also be adapted to obtain a collection of splay trees structure [12], which performs the three dynamic LMPT operations in O(log n) amortized time and which adapts to provide faster lookups for bursty traffic.

Lu and Sahni [13] show how priority search trees may be used to support the three dynamic router-table operations in O(log n) time each. Their work applies to both prefix filters as well as to the case of a set of conflict-free range filters.

2.3 Early Rejection Rules

The technique of early rejection rules was proposed by Adel El-Atawy et al. [14], using an approximation algorithm that analyzes the firewall policy in order to construct a set of early rejection rules. This set can reject the maximum number of unwanted packets (discarded by the policy rules) as early as possible. Unfortunately, the construction of the optimal set of rejection rules is an NP-complete problem and adding them may increase the size of matching rules list.

This technique uses an approximation algorithm that pre-processes the firewall policy off-line and generates different near optimal solutions.

This early rejection technique works periodically by building a list of most frequently hit rejection rules based on the network traffic statistics. And then, starts comparing the incoming packets to that list prior to trying the normal packet filter rules.

3. Proposed Work

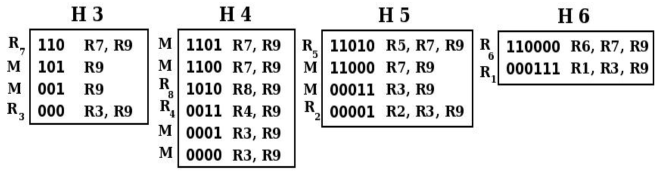
Our proposed scheme is a set of self-adjusting filters and it is called SA-BSPL: Self-Adjusting Binary Search on Prefix Length. In this work, we perform the scheme of Waldvogel et al [1] by using the splay tree model to ameliorate the average search time.

Our proposed filter can be applied to filter every packet's header field. Accordingly, it can easily assure exact matching for protocol field, prefix matching for IP addresses, and range matching for port numbers.

Generally, the process of packet classification is based on more than one packet header field of the incoming packet. Especially for range fields, we have to use the direct range-to-prefix conversion algorithm to transform each individual range into one or more prefixes.

The binary search on prefix length algorithm [1] is known to be efficient in search performance and it is based on three main ideas:

First, it uses hashing to check whether an incoming value matches any prefix of a particular length; second, it uses binary search to reduce number of searches from linear to logarithmic; third, it uses pre-computation to prevent backtracking in case of failures in the binary search.



H 3

R_7	110	R7, R9
M	101	R9
M	001	R9
R_3	000	R3, R9

H 4

M	1101	R7, R9
M	1100	R7, R9
R_8	1010	R8, R9
R_4	0011	R4, R9
M	0001	R3, R9
M	0000	R3, R9

H 5

R_5	11010	R5, R7, R9
M	11000	R7, R9
M	00011	R3, R9
R_2	00001	R2, R3, R9

H 6

R_6	110000	R6, R7, R9
R_1	000111	R1, R3, R9

Figure 3. The collection of hash-tables according to the destination address field of Table. 1

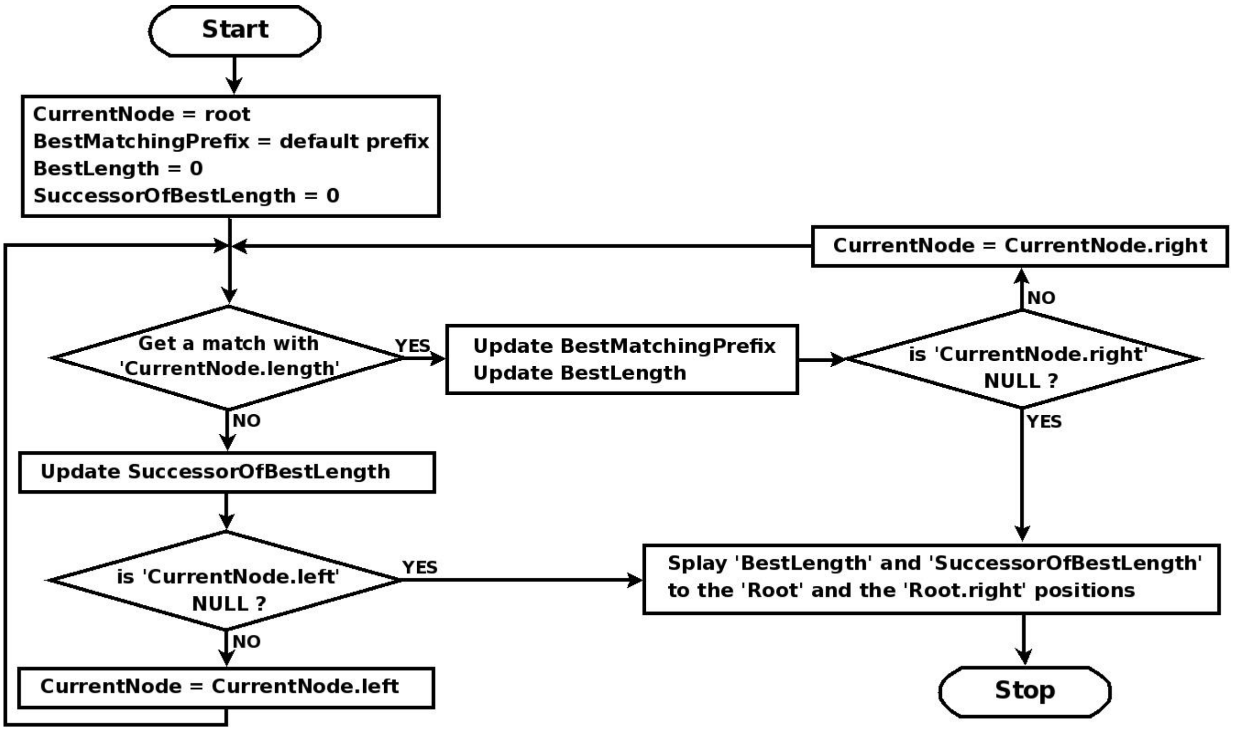

Start
CurrentNode = root
BestMatchingPrefix = default prefix
BestLength = 0
SuccessorOfBestLength = 0
CurrentNode = CurrentNode.right
NO
Get a match with 'CurrentNode.length'
YES
Update BestMatchingPrefix
Update BestLength
is 'CurrentNode.right' NULL ?
YES
NO
Update SuccessorOfBestLength
is 'CurrentNode.left' NULL ?
YES
Splay 'BestLength' and 'SuccessorOfBestLength' to the 'Root' and the 'Root.right' positions
NO
CurrentNode = CurrentNode.left
Stop

Figure 4. Proposed algorithm for determining and splaying of the best length value and its successor while searching on prefix lengths.

In order to ameliorate the scheme of Waldvogel, our work is based on three main optimization techniques:

- We adapt the splay tree model to the binary search on prefix length algorithm to filter every packet header field and to take into consideration the skewness in the traffic.
- We give a special interest in the amount of packets treated then rejected through the default entry because these flows might cause more harm than others as they traverse a long decision path before they are finally sent to the default entry.
- With the use of the top-down splay tree, we can reach a better classification time because we are able to combine searching and restructuring steps together.

3.1 Splay Tree Filters

The proposed scheme consists of a collection of hash-tables and a splay tree with no need to represent the default prefix (with zero length).

On one hand, the prefixes are grouped by lengths in hash-tables, as illustrated on the Figure 3. Also, each hash-table is augmented with markers for longer length prefixes (Figure. 1 shows an example).

On the other hand, the different values of lengths are stored in a splay tree which is an efficient implementation of binary search trees that takes advantage of locality in the incoming lookup requests.

Locality in this context is a tendency to look for the same element multiple times. Using splay trees, nodes that are often accessed will reside close to the tree root. Hence, we reduce dynamically the number of memory accesses to reach quickly the required result.

In our scheme, we still use binary search on prefix length but with splaying operations to push every newly accessed node to the top of the tree and because of the search phase starts at root and ends at the leaves of the splay tree, we have to splay the best length value to the root and its successor to the root.right position (Figure 5).

The trivial composed splaying operation (CSplay) is expressed as follows:

$$CSplay(x, x+) = Splay(x, root) + Splay(x+, x.right) \quad (1)$$

Consequently, the tree is adequately adjusted to have at most 2 hash accesses for all repeated values. We start the search process from the root node of the splay tree and if we get a successful match in the corresponding hash-table, we update the best length value and the best matching prefix then we go for higher lengths and if nothing is matched, we update the successor of the best length value then we go for lower lengths (Figure 4). We stop the search process if a leaf is met. After that, the best length value and its successor have to be splayed to the top of the tree.

In this case, we can use either bottom-up or top-down splay trees. However, the top-down splay tree is much more efficient because we are able to combine searching and splaying stages together.

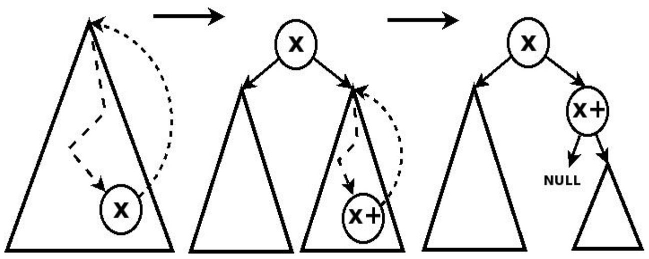


Figure 5. The operation of splaying of the item x and its successor (We assume that x is the best length value and x+ its successor).

3.2 Alternative Optimized Technique

In this subsection, we are going to give a better alternative implementation of the splaying operations. In the other words, the best length value and its successor are to be splayed to the top of the tree with a slightly better amortized time bound than the trivial one given in Figure 5.

We assume that the item x is the best length value and x+ its successor. When we search for the best length value, we start searching from the root node until we reach a leaf. The item x and x+ belong to the same search path as it is shown in Figure 6, Figure 7 and Figure 9. Hence, we have two alternatives for this case.

Figure 6 shows that the best length x may appear before its successor x+ in the search path. In this case, we have to splay x+ until it becomes the right child of x. Then we splay x and x+, as a single node, to the top of the tree.

The optimized composed splay operation (OCSplay) is expressed as follows (2):

$$OCSplay(x, x+) = Splay(x+, x.droit) + Splay((x, x+), root) \quad (2)$$

The elementary operation Splay(x_1, x_2) means that the item x_1 has to be splayed to the top of the tree which root node is x_2.

Figure 7 shows that the best length x may appear after its successor in the search path. Hence, we have to splay x until it becomes the parent of x+ then these two nodes have to be splayed as single node to the top of the tree. In this case, the optimized composed splay operation (OCSplay) is expressed as follows (3):

$$OCSplay(x, x+) = Splay(x, x+) + Splay((x, x+), root) \quad (3)$$

Consequently, we have a better amortized cost of the composed splaying operations and we are going to give a proof for the logarithmic amortized cost of these operations in the next section.

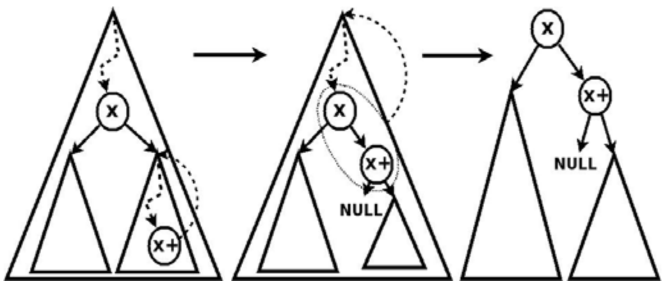


Figure 6. Splaying x and x+ as a single node (x appears before x+ at the search path).

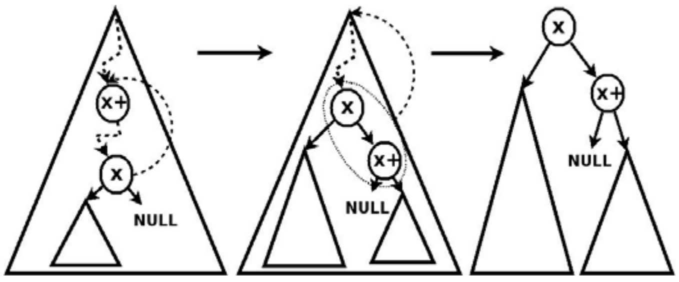


Figure 7. Splaying x and x+ as a single node (x appears after x+ at the search path).

3.3 Range to Prefix Conversion

As we saw before, such a matching paradigm works well for prefix matching of IP addresses but is not well-suited to matching fields with ranges (e.g., port number range).

Packet classification involves various matching conditions, e.g., longest prefix matching (LPM), exact matching, and range matching, making it a complicated pattern matching issue. [15] reported that today's real-world policy filtering (PF) tables involve significant percentages of rules with ranges. The transport-layer fields have a wide variety of specifications. Many (10.2%) of them are range specifications (e.g. of the type gt 1023, i.e. greater than 1023, or in range 20-24). In particular, the specification 'gt 1023' occurs in about 9% of the rules.

The usual way is to convert each range into a set of prefixes. With the use of the direct range-to-prefix conversion, each range is individually converted into one or more prefixes. An efficient direct range-to-prefix conversion algorithm can be found in [16] and [17].

For example, the range R = [2, 6] is converted into three prefixes, 001*, 010*, and 0110. In the worst case, the range [1, 2^W – 2] is split into 2W – 2 prefixes. For a set of m ranges, the worst-case number of prefixes generated by a direct range-to-prefix conversion algorithm is O(mW) [17] with W is the maximum number of bits in the address space of prefixes or ranges.

3.4 Minimizing the Number of Tree Rotations

In order to reduce the number of tree rotations while adjusting the splay tree, we have to maintain each assembled values (best length value and its successor) linked together. The attached nodes have to be moved as several single nodes; therefore we have to put solid edges between the best length values and their successors and normal edges between the other nodes (Figure 8).

The only case where the solid edge between x and x+ have to be broken is when the successor of the best length value x+ is the node to be rotated to the top of the tree.

The resulting tree constructed with solid and normal edges or with single and double nodes is able to maintain all the last accessed pairs of node close to the top of tree to be quickly found.

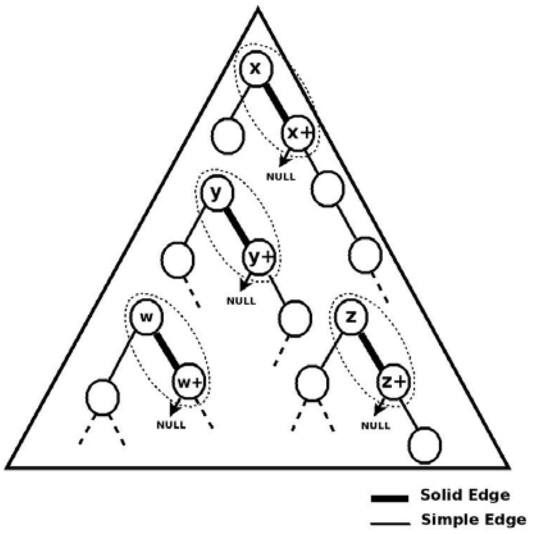


Figure 8. An efficient technique to minimize tree rotations.

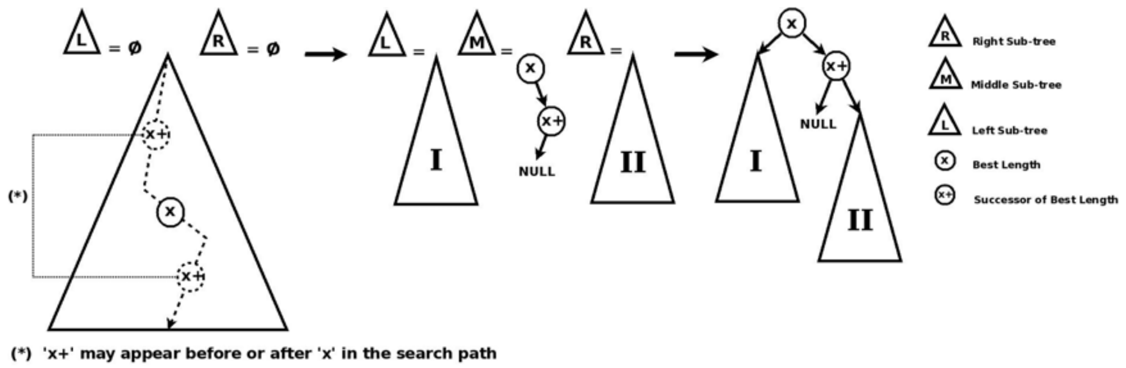


Figure 9. Searching process in the Top-Down Splay Tree.

3.5 Top-Down Splay Tree

Various variants, modifications and generalization of splay trees have been studied; we give for example [18] and [19]. The top-down splay trees [8] are another way of implementing splay trees.
On one hand, the top-down algorithm is much more efficient than the bottom-up because we are able to combine the search and the tree reorganization into a single phase (Figure 9). On the other hand, it is the method recommended by the inventors of splay trees as a means of avoiding having to traverse the search path again. Both the bottom-up and the top-down splay trees coincide if the node being searched is at an even depth [19], but if the item being searched is at an odd depth, then the top-down and bottom-up trees may differ [18] and some experimental evidence suggests [20] that top-down splay trees [8], [19] are faster in practice as compared to the normal splay trees.
We have adapted the binary search on prefix length algorithm to the top-down splay tree as given in Figure 9 to improve the performance of the bottom-up algorithm given (Figure 4). In top-down splay trees, we look at two nodes at a time, while searching for the best length value, and also we keep restructuring the tree until we reach a leaf and the item we are looking for has been located.
When searching for the best length value, the current tree is divided into three sub-trees, while we move down two nodes at a time searching for the query item.

- Middle Sub-tree: This sub-tree is continuously updated while we move down two nodes at a time searching for the best length value in the original tree and it contains only two nodes (x and x+) and if we update the best length and its successor while searching, we insert the old values of x and x+ respectively in the left sub-tree and the right sub-tree.
- Left Sub-tree: When we get a successful match with the current item we put it in the left sub-tree. The left sub-tree contains items which are smaller than the item we are searching.
- Right Sub-tree: When we don't get match with the current node we put it in the right sub-tree. The right sub-tree consists of items which are larger than the item we are searching.

Finally, the sub-trees have to be linked to each other to form the new tree which becomes ideally adjusted to have at most 2 hash accesses for all the repeated values.

3.6 Early Rejection Technique

In this section, we focus on optimizing matching of traffic discarded due to the default-deny rule because it has more profound effect on the performance of the firewalls.
In our case, we have no need to represent the default prefix (with zero length) so if a packet don't match any length it will be automatically rejected by the Min-node. Generally, rejected packets might traverse long decision path of rule matching before they are finally rejected by the default-deny rule. The left child of the Min-node is Null, hence if a packet doesn't match the Min-node we go to its left child which is Null, so it means that this node is the end of the search path.
In our case, in each filter, we have to traverse the entire tree until we arrive to the node with the minimum value. Subsequently, a packet might traverse a long path in the search tree before it is rejected by the Min-node. Hence, we have to rotate always the Min-node in the upper levels of the self-adjusting tree. We have to splay the Min-node to the root.left position as shown in (Figure 10).
In our case, we can use either bottom-up or top-down splay trees. However, the top-down splay tree is much more efficient for the early rejection technique because we are able to maintain the Min-node fixed at the desired position when searching for the best matching value without explicitly splaying it.

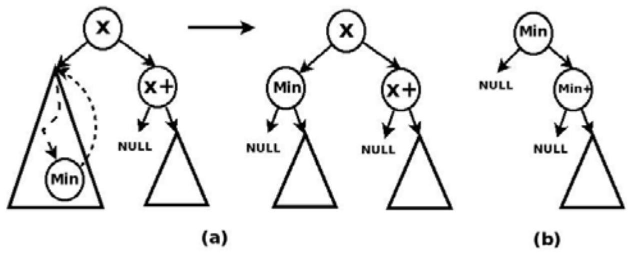


Figure 10. Maintaining the Min-node at the upper level of the search tree (Bottom-Up Splay Tree)

4. Complexity Analysis

In this section, we first give the complexity analysis of the proposed operations used to adapt the splay tree data structure to the binary search on prefix length algorithm and we also give the cost per filter of our early rejection technique. Then, we compare our splay tree packet classification scheme SA-BSPL with the ST-PC scheme described in Section 2 in term of space usage and search time.

4.1 Amortized Analysis

Splay trees are a simple and efficient data structure for storing an ordered set. It consists of a binary tree, with no additional fields and it allows searching, insertion, deletion, splitting, joining, and many other operations, all with amortized logarithmic performance.

According to the analysis of Sleator and Tarjan [8], if each item of the splay tree is given a weight w_x, with W_t denoting the sum of the weights in the tree t, then the amortized cost to access an item x have the following upper bounds (Let x+ denote the item following x in the tree t and x- denote the item preceding x):

$$3\log\left(\frac{W_t}{w_x}\right) + O(1) \text{ if x is in the tree t} \quad (4)$$

$$3\log\left(\frac{W_t}{\min(w_{x-}, w_{x+})}\right) + O(1) \text{ if x is not in the tree t} \quad (5)$$

In our case, we have to rotate the best length value to the root position and its successor to the root.right position (Figure 5). We have a logarithmic amortized complexity to release these two operations and the time cost is calculated using (4) and (5):

$$3\log\left(\frac{W_t}{w_x}\right) + 3\log\left(\frac{W_t - w_x}{w_{x+}}\right) + O(1) \quad (6)$$

With the use of the alternative optimized implementation of splaying (Figure 6 and Figure 7) we obtain slightly better amortized time bound than the one given in (6):

$$3\log\left(\frac{W_t - w_x}{\min(w_x, w_{x+})}\right) + O(1) \quad (7)$$

On the other hand, the cost of the early rejection step (Bottom up case) is:

$$3\log\left(\frac{W_t - w_x}{w_{\min}}\right) + O(1) \quad (8)$$

We note that $w_{\min}$ is the weight of the Min-node. With the use of the proposed early rejection technique, we have at most 2 hash accesses before we reject a packet and at least only one. If we have m consecutive values to be rejected by the default deny rule, search time will be at most m+1 hash accesses per filter and at least m hash accesses.

4.2 Number of Nodes

Space and time complexity are related to the number of nodes in the search tree. Our self-adjusting scheme is based on the binary search on prefix length scheme which is a collection of hash-tables organized by prefix lengths. Hence, the number of nodes in our case is equal to the number of hash-tables. Subsequently, if we consider W the length of the longest prefix in the packet filter list, we have at most w nodes (Figure 11).

$$n_{SA-BSPL} = w \quad (9)$$

For the ST-PC scheme, if we assume that all the prefixes are distinct, then we have at most 2^W nodes in the worst case. Besides, if all bounds are distinct, then we have 2r nodes, we note that r is the number of entries in the packet classifier list. So, the actual number of nodes in ST-PC in the worst case will be the minimum of these two values.

$$n_{ST-PC} = \min(2^w, 2r) \quad (10)$$

Since the number of nodes in our self-adjusting tree is smaller than in the ST-PC especially with an important number of filtering rules (Figure 11), we can say that our splay tree scheme is much more competitive in term of time and scalability.

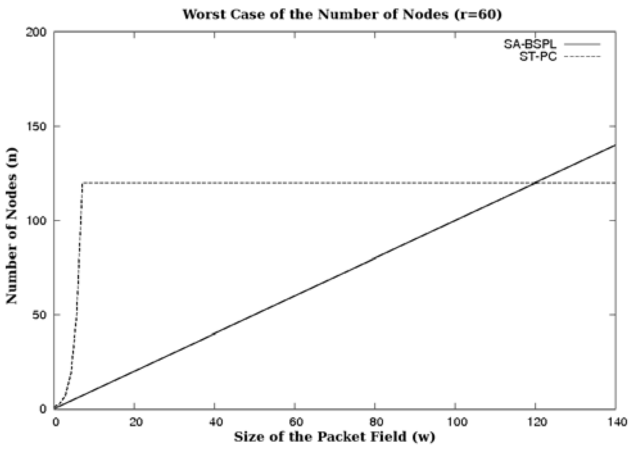


Figure 11. This figure shows the distribution of the number of nodes with respect to w and r, where w is the maximum possible length in bits and r the number of rules.

4.3 Memory accesses

Memory accesses are expensive and are the dominant factor in determining the worst-case execution time and it's the most costly operation in the packet classification schemes. When analyzing the performance of a given algorithm, a common technique is to differentiate the average and the worst case performance of a resource usage. Usually the resource being considered is the running time.

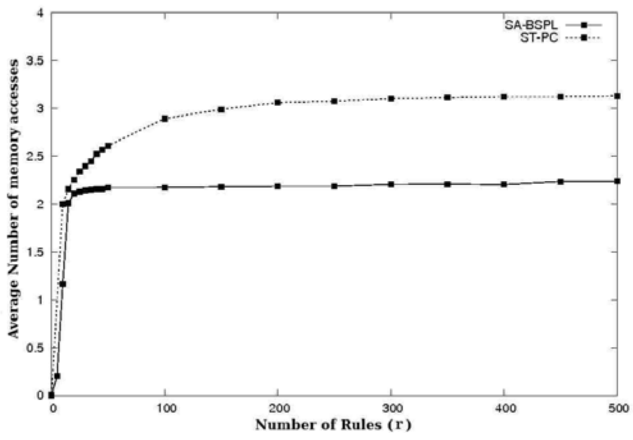


Figure 12. The increase of the average number of memory accesses with the increase of the number of rules.

Figure 12 compares the average number of memory accesses required for both SA-BSPL and ST-PC techniques for an arbitrary sequence of incoming values. Here again, the performance of the proposed filter is better than the ST-PC technique.
The worst case time behavior can also be viewed through the distribution of the worst case of the number of nodes in the search tree (Figure 11).

5. Conclusion

The packet classification optimization problem has received the attention of the research community for many years. Nevertheless, there is a manifested need for new innovative directions to enable filtering devices such as firewalls and QoS routers to keep up with the high-speed networking demands. In our work, we have suggested a dynamic scheme, based on collections of hash-tables and splay trees to filter the incoming packets based on the destination and source IP addresses. The proposed scheme can be easily adapted to filter the other header fields of the incoming packets. We have also proved that the proposed model is suitable to take advantage of the locality in the incoming requests and it outperforms the previous techniques especially for highly skewed incoming values. In conclusion, we have reached the desired goal with a logarithmic time cost, and in our future works, we wish to optimize data storage and the other performance aspects.

References

[1] M. Waldvogel, G. Varghese, J. Turner, B. Plattner. "Scalable High Speed IP Routing Lookups". In *Proceedings of the ACM SIGCOMM (SIGCOMM '97)*, pp. 25-36, 1997.

[2] H. Lim, J. Hyoung. "High-speed Packet Classification using Binary Search on Length". In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS'07)*, pp. 137-144, 2007.

[3] H. Lu, S. Sahni. "O(logW) Multidimensional Packet Classification". *IEEE/ACM Transactions on Networking (TON)*, 15 (2), pp. 462-472, 2007.

[4] T. Srinivasan, M. Nivedita, V. Mahadevan. "Efficient Packet Classification Using Splay Tree Models". *IJCSNS International Journal of Computer Science and Network Security*, 6(5), pp. 28-35, 2006.

[5] V. Srinivasan, G. Varghese. "Fast Address Lookups using Controlled Prefix Expansion". *ACM Transactions on Computer Systems (TOCS'99)*, 17(1), pp. 1-40, 1999.

[6] K. Kim, S. Sahni. "IP Lookup by Binary Search on Prefix Length". In *Proceedings of the 8th IEEE International Symposium on Computers and Communications (ISCC'03)*, pp. 77-82, 2003.

[7] A. Broder, M. Mitzenmacher. "Using Multiple Hash Functions to Improve IP Lookups". In *Proceedings of the IEEE INFOCOM*, pp. 1454–1463, 2001.

[8] D. Sleator, R. Tarjan. "Self Adjusting Binary Search Trees". *Journal of the ACM*, 32(3), pp. 652-686, 1985.

[9] P. Gupta, B. Prabhakar, S. Boyd. "Near-optimal Routing Lookups with Bounded Worst Case Performance". In *Proceeding of IEEE INFOCOM*, pp. 1184-1192, 2000.

[10] F. Ergun, S. Mittra, S. C. Sahinalp, J. Sharp, R. K. Sinha. "A dynamic lookup scheme for bursty access patterns". In *Proceeding of IEEE INFOCOM*, pp. 1444-1453, 2001.

[11] S. Sahni, K. Kim. "O(log n) Dynamic Packet Routing". In *Proceedings of the 7th International Symposium on Computers and Communications (ISCC'02)*, pp. 443-448, 2002.

[12] S. Sahni, K. Kim. "Efficient Dynamic Lookup for Bursty Access Patterns". In *http://www.cise.ufl.edu/~sahni*, 2003.

[13] H. Lu, S. Sahni. "O(log n) Dynamic Router-tables for Prefixes and Ranges". In *IEEE Transactions on Computers*, 53(10), 2004.

[14] H. Hamed, A. El-Atawy, E. Al-Shaer. "Adaptive Statistical Optimization Techniques for Firewall Packet Filtering". In *Proceeding of IEEE INFOCOM,* pp. 1-12, 2006.

[15] E. Spitznagel, D. Taylor, J. Turner. "Packet Classification Using Extended TCAMs". In *Proceedings of the 11th IEEE International Conference on Network Protocols*, pp.120-131, 2003.

[16] M.D. Berg, M.V. Kreveld, M. Overmars, O. Schwarzkopf. "Computational Geometry: Algorithms and Applications", Springer Verlag, 1997.

[17] Y.K. Chang, "A 2-Level TCAM Architecture for Ranges", In *IEEE Transactions on Computers*, 55(12), pp. 1614-1629, 2006.

[18] S. Albers, M. Karpinkski. "Randomized Splay Trees: Theoretical and Experimental Results". *Information Processing Letters*, 81(4), pp. 213-221, 2002.

[19] E. Makinen. "On Top-down Splaying". *BIT Computer Science and Numerical Mathematics*, 27(3), pp. 330-339, 1987.

[20] J. Bell, G. Gupta. "An Evaluation of Self-adjusting Binary Search Tree Techniques". *Software-Practice and Experience*, 23(4), pp. 369-382, 1993.

Author Biographies

Nizar Ben Neji received in 2005 his engineering degree in computer science from the national Tunisian school of computer sciences (ENSI) and he received in 2008 his MS degree from the higher school of communications of Tunis (Sup'Com). Nizar Ben Neji is actually a PhD student at Sup'Com and his research interests are about security devices and optimizing the performance of the packet classifiers used in high speed networks. Nizar BEN NEJI works also as computer engineer in the National Digital Certification Agency (NDCA: The root certification authority in Tunisia) and he is a member of the Tunisian Association of Digital Security (TADS).

Adel Bouhoula obtained his undergraduate degree in computer engineering with distinction from the University of Tunis in Tunisia. He also holds a Master's, PhD and Habilitation from Henri Poincare University in Nancy, France. Adel Bouhoula is currently an Associate Professor at the Sup'Com Engineering School of Telecommunications in Tunisia. He is also the founder and Director of the Research Unit on Digital Security and the President of the Tunisian Association of Digital Security (TADS). His research interests include Automated Reasoning, Algebraic specifications, Rewriting, Network Security, Cryptography, and Validation of cryptographic protocols.