

# Supervised Grid-of-Tries: A Novel Framework for Classifier Management

Srinivasan T.<sup>1</sup>, Balakrishnan R., Gangadharan S.A., and Hayawardh V.

<sup>1</sup> Assistant Professor, Department of Computer Science and Engineering,  
Sri Venkateswara College of Engineering, Sriperumbudur, TN, India 602 105  
tsrini@svce.ac.in, bsrealm@msn.com,  
{gangadharan, hayawardh}@gmail.com

**Abstract.** Packet classification is the problem of identifying which one of a set of rules maintained in a database is best matched by an incoming packet at a router and taking the action specified by the rule. This is a uniform enabler for many new network services like firewalls, quality of service and virtual private networks. These services require dynamic management of rules. While many algorithms recently proposed can perform packet classification at very high speeds, rule update times for these are not very fast. This paper presents an efficient classifier management algorithm, which effectively reduces the rule update time for the well known Grid-of-Tries classifier. To this end, we have devised a novel structure called Supervised Grid-of-Tries, which employs additional tracking pointers embedded into the trie to facilitate efficient rule updates.

**Keywords:** Packet classification, routing, grid-of-tries, supervised grid-of-tries.

## 1 Introduction

Packet classification is the underlying mechanism facilitating network services like quality of service, virtual private networks (VPN) and firewalls.

Several approaches to packet classification have been proposed. Traditional trie-based approaches to packet classification include the Hierarchical Trie, Set Pruning Trie and Grid-of-Tries with filter update complexities of  $O(dw)$ ,  $O(n^d)$  and  $O(nw)$  respectively, where  $d$  denotes the number of dimensions,  $n$  the number of nodes and  $w$  the width of each dimension. This does not scale up well on large filter sets. Here, we present a structure that performs updates efficiently, even for large filter sets.

The paper is organized as follows. Section 2 describes the preliminaries. Section 3 portrays our proposed approach. Experimental results are displayed in Section 4. In Section 5, guidelines to choose between the schemes discussed in Section 3 are suggested. Concluding remarks are in Section 6.

## 2 Preliminaries

Here, the term “ancestor trie” refers to any trie which is under a first dimension node that is an ancestor of the first dimension node under which the current trie is present.

The term “lowest ancestor trie” refers to the most immediate ancestor trie. In addition, we employ “links” to mean switch pointers and/or storedFilters.

*Supervision Tree of Tries:* A multi way tree representing the dependency hierarchy amongst the tries in the second dimension.

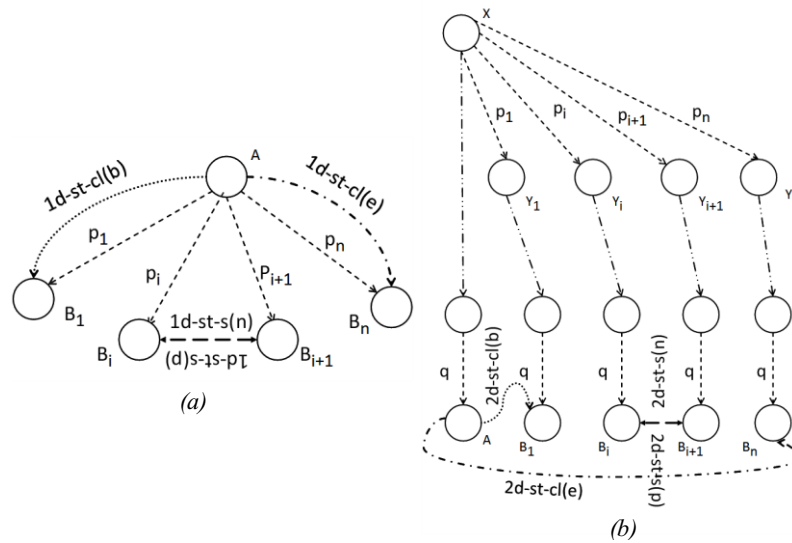
*Supervision Tree of Nodes:* A multi way tree representing the dependency hierarchy amongst the nodes in the second dimension.

Let  $A$  be a first dimension node (Fig. 1a). Let  $S = \{B_1, B_2, \dots, B_n\}$  be the set of nodes in  $A$ 's sub tree reachable along paths  $p_1, p_2, \dots, p_n$  respectively, such that

1.  $B_i$  has a trie under it for  $1 \leq i \leq n$
2. No node in path from  $A$  to  $B_i$  has a trie under it for  $1 \leq i \leq n$
3.  $p_i < p_{i+1}$  (lexicographically) for  $1 \leq i \leq n - 1$
4.  $B_i \neq A$  for  $1 \leq i \leq n$

Then,  $S$  is the set of first dimension ( $1d$ )  $st$  children of  $A$  and there exists

1. A  $1d$  supervised trie child list begin ( $1d$ - $st$ - $cl$ (b)) link from  $A$  to  $B_1$ .
2. A  $1d$  supervised trie child list end ( $1d$ - $st$ - $cl$ (e)) link from  $A$  to  $B_n$ .
3. A  $1d$  supervised sibling next ( $1d$ - $st$ - $s$ (n)) link from  $B_i$  to  $B_{i+1}$  for  $1 \leq i \leq n - 1$
4. A  $1d$  supervised sibling previous link from  $B_{i+1}$  to  $B_i$  for  $1 \leq i \leq n - 1$ .



**Fig. 1.** (a)  $st$  First dimension child and sibling pointers. (b)  $st$  Second dimension child and sibling pointers.

Let  $A$  be a second dimension node (Fig. 1b) reached from the root of its trie along edges labelled  $q$ . Let this trie be under node  $X$  in the first dimension. Let  $S' = \{ Y_1, Y_2, \dots, Y_n \}$  be the set of nodes in  $X$ 's sub tree reached along paths  $p_1, p_2, \dots, p_n$  respectively such that

1.  $Y_i$  has a trie under it that has a node reachable along edges labelled  $q$  for  $1 \leq i \leq n$
2. No node from  $A$  to  $Y_i$  has a trie under it which has a node reachable along edges labelled  $q$  for  $1 \leq i \leq n$
3.  $p_i < p_{i+1}$  (lexicographically) for  $1 \leq i \leq n - 1$
4.  $Y_i \neq A$  for  $1 \leq i \leq n$

Let  $S = \{B_1, B_2, \dots, B_n\}$  be the set of nodes such that  $B_i$  is reached from the root of the trie under  $Y_i$  along edges labelled  $q$  for  $1 \leq i \leq n$ . Then,  $S$  is the set of second dimension ( $2d$ ) *st* children of  $A$  and there exists

1. A  $2d$  supervised trie child list begin ( $2d$ -st-cl(b)) link from  $A$  to  $B_1$ .
2. A  $2d$  supervised trie child list end ( $2d$ -st-cl(e)) link from  $A$  to  $B_n$ .
3. A  $2d$  supervised sibling next ( $2d$ -st-s(n)) link from  $B_i$  to  $B_{i+1}$  for  $1 \leq i \leq n-1$ .
4. A  $2d$  supervised sibling previous ( $2d$ -st-s(p)) link from  $B_{i+1}$  to  $B_i$  for  $1 \leq i \leq n-1$

In the next section, we endeavor to present schemes resulting in efficient filter update for the Grid-of-Tries.

### 3 Proposed Approach

When inserting a rule into the Grid-of-Tries, there may be a need to set links from tries further below to the newly inserted trie or to have switch pointers or storedFilters emanating from the inserted trie itself to a trie above it. Unless we have a systematic method to track which tries need to be updated as a result of inserting or deleting a rule, it is tricky to perform incremental updates. We attempt to save on rule update time by modifying the existing Grid-of-Tries instead of complete reconstruction. During rule update, the Grid-of-Tries is modified in the first and second dimensions based on the rule. All other tries require amendment only if their lowest ancestor also does. Thus, there is a dependency hierarchy present among the tries and nodes. This hierarchy can be structured in the form of a supervision tree.

#### 3.1 One Dimension Supervised (*1ds-SGOT*)

*1ds-SGOT* implements only the supervision tree of tries. In this algorithm, the first dimension nodes which have tries under them act as representatives of those tries. In order to insert a rule, we first traverse the first dimension. If a new node was created, we backtrack and update the supervision tree of tries. Next, the second dimension trie is traversed. If any new nodes were created in the second dimension, we set links from the current trie to the ancestor tries and to the current trie from tries in its *st* sub tree as necessary.

The worst case time complexity of this algorithm is  $O(nw)$ . However, occurrence of the worst case requires that several rare conditions be satisfied. Also, once the worst case has occurred, it cannot occur again until the rule that caused the worst case is removed. Memory requirements double for the first dimension in comparison to the Grid-Of-Tries algorithm due to the four additional pointers (*st*) that have to be maintained in addition to the existing four in each node.

### 3.2 Two Dimension Supervised (*2ds-SGOT*)

Performance can be further improved in certain environments at the expense of memory. The *2ds-SGOT* algorithm implements the supervision trees of nodes in addition to the supervision tree of tries.

The first dimension is handled as in *1ds-SGOT*. In the second dimension, if any new nodes are created, we set the *st* sibling and *st* child node pointers as appropriate. We then set switch pointers and storedFilters to corresponding nodes in the ancestor tries and to the current nodes from nodes in the sub tree of this node's supervision tree of nodes.

In *2ds-SGOT*, repeated access of nodes in the second dimension trie of the rule (that took place in *1ds-SGOT*) is avoided. All operations on a node are finished in a single visit. Also, nodes that do not require an update are skipped. Hence we stand to gain a reduction in worst case time, while the worst case complexity and conditions remain unchanged. In order to support the above features, extra processing is required through the maintenance of *st* pointers in the second dimension nodes. Memory requirements for the second dimension are approximately twice that of *1ds-SGOT* whereas the memory consumption of the first dimension is the same. Rule deletion can be performed along the same lines as insertion. With some modifications, the same approach can be used to perform supervised rule updates on extended grid of tries.

## 4 Experimental Results

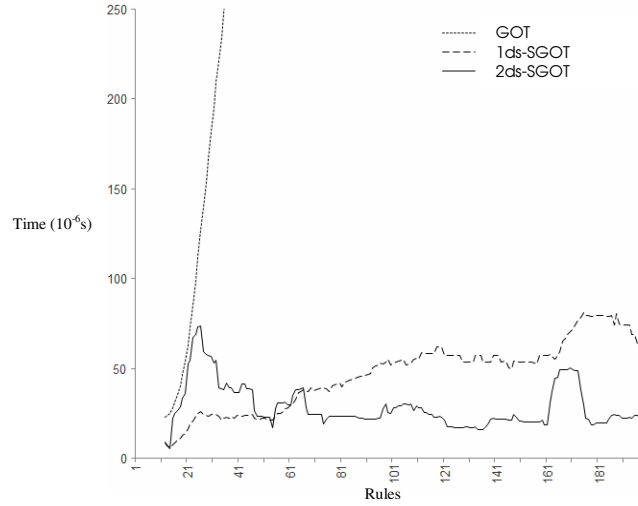
The performance of *1ds-SGOT* is significantly dependent on the number of tries visited for updating. However, this is not the case with *2ds-SGOT* since, as stated previously, it avoids the access of nodes which are certain to not require updates. Besides, the number of times these nodes are visited is one as compared to the multiple times they are visited in *1ds-SGOT*.

We now proceed to analyze and compare the performance of our proposed algorithms against each other and the Grid-of-Tries method in different environments. The filter sets we utilize for our analysis are drawn from [4].

Our empirical results confirm that the time consumed by the conventional method for a rule update increases linearly with the number of rules (as the structure is completely reconstructed from the beginning with every new rule added) in the trie whereas it remains nearly constant for *1ds-SGOT*.

*2ds-SGOT* performs best in firewalls, which are most specific (and hence have the most number of tries), which is in conformance with the above deductions. Also, as expected, *1ds-SGOT* performs better than *2ds-SGOT* in the least specific environment of access control lists.

Table 2 shows the average update time for our algorithms on the various filter sets [6]. It is clear from the table that, as the specificity increases, the *2ds-SGOT* algorithm performs better than the *1ds-SGOT* algorithm.



**Fig. 2.** Rule insertion times of the algorithms for sample filter set FW1

**Table 2.** Performance Comparison. Filter sets taken from [4].

Filter set	1ds-SGOT ( $10^{-6}$ s)	2ds-SGOT ( $10^{-6}$ s)
ACL1	41	61
ACL1_100	48	133
ACL1_1K	47	76
ACL1_5K	53	68
ACL1_10K	63	98
IPC1	69	51
IPC1_100	47	129
IPC1_1K	57	100
IPC1_5K	88	67
IPC1_10K	155	74
FW1	45	33
FW1_100	28	39
FW1_1K	109	38
FW1_5K	445	80
FW1_10K	1054	120

## 5 Selection of Optimal Scheme

We now discuss metrics that help in selecting the optimal scheme.

*Specificity:* We infer from the experimental results that as the specificity of the filter sets increases, the relative performance of *2ds-SGOT* improves over *1ds-SGOT* due to the increased number of tries. Thus it would be more advantageous to use *2ds-SGOT* in environments like firewalls. Conversely, *1ds-SGOT* should be preferred in environments like access control lists.

*Scalability:* With the impending transition to IPv6, the relative performance benefit for *2ds-SGOT* over *1ds-SGOT* would be amplified due to the significant increase in node accesses for rule updates. With increase in the number of rules, *2ds-SGOT* performs better than *1ds-SGOT*.

*Reliability:* In situations where the memory consumption of *2ds-SGOT* is about to exceed the available memory, there can be a seamless transition to *1ds-SGOT* which will enable the router to support as many rules as can *1ds-SGOT* while at the same time providing the performance of *2ds-SGOT* until no longer possible.

## 6 Conclusion

We have devised a novel method for efficient dynamic filter update for the Grid-of-Tries classifier. This is achieved by maintaining a supervision trees to track those parts of the trie which require updates. Through our experimentation, it is shown that our two techniques have nearly constant filter update times, whereas the conventional method has an update time which increases linearly with the number of filters.

## References

1. V. Srinivasan, S. Suri, G. Varghese and M. Waldvogel. "Fast and scalable layer four switching," Proceedings of ACM Sigcomm, pages 203-14, September 1998.
2. T. Srinivasan, S. Prasad, B. Prakash, "Dynamic Packet Classification Algorithm using Multi-level Trie", Enformatika, Volume 3, pp.104-107, Transactions on Engineering, Computing and Technology, Dec 2004 (ISSN 1305-5313).
3. T. Srinivasan, Dhanasekar, M. Nivedita, B. Divya, Azeezunnisa Shakir "Scalable and Parallel Aggregated Bit Vector packet classification using prefix computation model", To appear in the proceedings of IEEE International Symposium on Parallel Computing in Electrical Engineering - PARELEC 2006, Bialystok, Poland, September 2006.
4. David E. Taylor, Jonathan S. Turner, "ClassBench: A Packet Classification Benchmark", IEEE INFOCOM 2005
5. T. Srinivasan, Azeezunnisa Shakir, Vijayalakshmi "PAFBV: A Novel Parallel Aggregated and Folded Bit Vector Packet Classification Scheme for IPv6 Routers", to appear in the proceedings of 6th IEEE International Conference on Computer and Information Technology - CIT 2006, IEEE Computer Society, Seoul, Korea, September 2006
6. Filter sets at <http://www.arl.wustl.edu/~det3/ClassBench/>