Layered Cutting Scheme for Packet Classification

Yeim-Kuan Chang and Han-Chen Chen
Department of Computer Science and Information Engineering
National Cheng Kung University, Tainan, 701, Taiwan
{ykchang, p76981285}@mail.ncku.edu.tw

Abstract—**Packet classification is an important topic for high-speed routers nowadays. There are many packet classification algorithms based on decision tree like Hicuts, Hypercuts and Hypersplit. Because Hicuts and Hypercuts divides the rule sets by cutting the address space into equal-sized subspaces, their cutting efficiency is not good. Although Hypersplit proposed a good end-point-based cutting scheme, the resulting tree depth is still very high. In this paper, we propose a multi-dimensional cutting algorithm to significantly reduce the decision tree depth and a multi-layered scheme to dramatically reduce the usage of memory. Our experimental results show that the proposed layered scheme needs much less memory than Hypersplit for Firewall and IPC rule tables with a factor of 2 to 106 improvement while the proposed layered scheme needs a little more memory than Hypersplit for some of ACL tables. In addition, in terms of number of memory accesses, the proposed layered scheme and Hypercuts are better than Hicuts and Hypersplit for all tables while the proposed layered scheme is better than Hypercuts for ACL and Firewall tables. In terms of number of memory accesses, our layered cutting scheme and Hypercuts perform equally well for small rule tables. But, in larger rule tables, the proposed layered cutting scheme has better performance.**

I. INTRODUCTION

Packet classification has been studied widely, such as firewall, QoS, traffic control, and VPNs. There are numerous existing solutions [1, 2, 3, 12, 15, 16] in the literature and many of them use decision trees as the basic data structure to make the process of classifying packets fast [4, 5, 7, 9, 11, 12, 14]. One regulation of the decision trees is the node's key comparison for deciding how to divide and distribute rules from an internal node into the child nodes. According to this regulation, rules are divided into subgroups each of which is put into associated child node and the dividing process is continued until the number of rules stored in each leaf node is no more than a pre-defined bucket size. When an incoming packet enters the classification engine, the decision tree is traversed from root, by comparing the headers of the packet with the internal node key, to one of its child node. Finally, the traversal will reach a leaf and then the rules in the leaf are checked for finding matched rule with the highest priority.

Differentiating where a decision tree based packet classification algorithm is good stands on the following factors.

- Memory Storage: Memory is needed to store internal and leaf nodes and the rules in the buckets of the leaf nodes. To implement the proposed architecture on FPGA, the memory is a key point because the block RAM on FPGA is limited.
- Search speed: Search time includes the time to traverse decision tree and the time to sequentially search the rule bucket in the leaf nodes. The sequential search time of the rules in the leaf nodes depends on the bucket size while tree traversal time depends on the height of the decision tree.

Good packet classification algorithms need small memory storage and have a high search speed. Hypersplit proposed in [11] is good at memory usage. However, because the decision tree used by Hypersplit is binary, the height of decision tree is too high, making search speed downcast. Our proposed scheme solves this problem by using a novel cutting scheme to build a multi-way decision tree for reducing tree depth. Although our proposed scheme could also reduce memory usage in IPC and FW rule tables, we want to improve it further. By observations, we find out that rule duplication is a big problem making the memory increase explosively. Because the relationship in certain internal node's rule set, some rules may be separated inefficiently. To solve this problem, we propose a novel layered cutting scheme to build the decision tree by moving those duplicated rules to next layer, and we could traverse all layered decision trees in parallel.

The rest of the paper is organized as follows: Section II introduces the most related packet classification algorithms that are Hicuts, Hypercuts, and Hypersplit. The multi-dimensional cutting scheme is proposed in section III. In section IV, the layered optimization is proposed. We present our experimental results in section V. Finally we conclude our proposed scheme and bring up the future work in final section.

II. RELATED WORK

Packet classification could be solved in many ways. The easiest way is to search the rule table sequentially, and output the highest priority rule. Although this solution needs the minimal memory size but is very inefficient in query time. Therefore many researches improved the query time by proposing better data structures. Tree structure is a good solution to improve the query time. If the tree has N child nodes, traversing the tree only needs a search time of O(logN) instead of O(N) by the sequential search. So, tree algorithm is the main solution in packet classification. Grid of Trie [10] is a fundamental tree based solution for packet classification. It improves the hierarchical trie by using switch pointers to avoid backtracks and the rule duplications. But, Grid of Trie can not be easily extended to more than two fields, making the search performance slow. We will introduce the better decision tree based algorithms in the following paragraph.

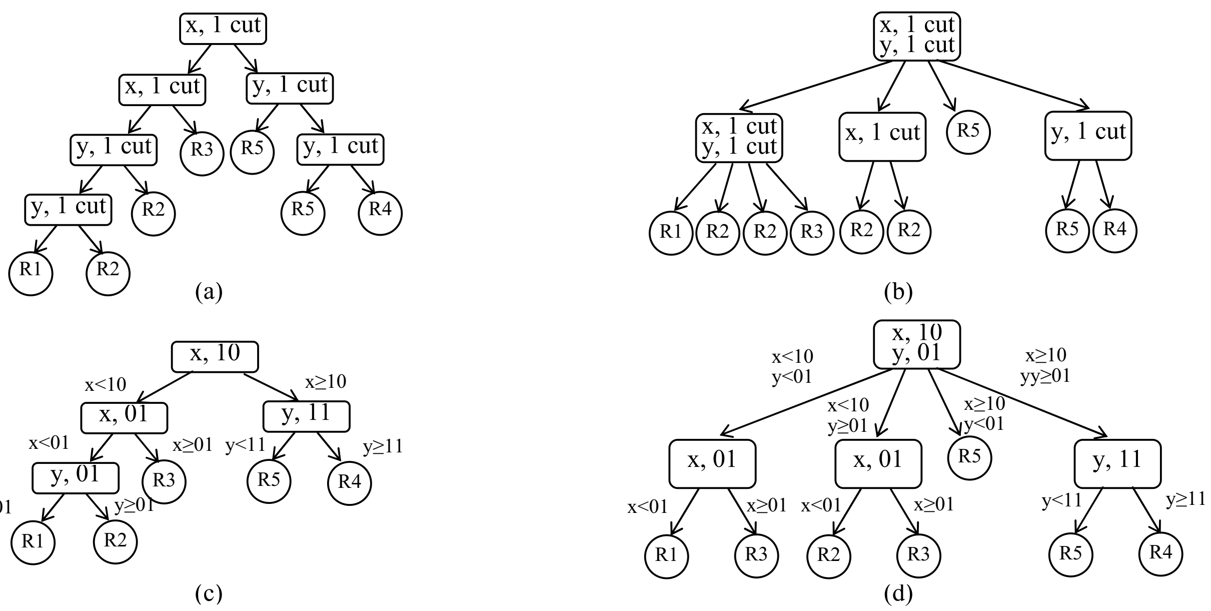


Figure 1. (a) Hicuts for the example rules. Need 6 internal node and 7 leaf, average tree height = 3.67 (b) Hypercuts for the example rules. Need 4 internal node and 9 leaf, average tree height = 2.89 (c) Hypersplit for the example rules. Need 4 internal node and 5 leaf, average tree height = 3.4 (d) Our algorithm for the example rules. Need 4 internal node and 7 leaf, average tree height = 2.86

Decision tree based packet classification algorithms focus on two aspects. The first one is how to select the cut dimension and the second is how to decide the cut-point for dividing address space into subspaces. There are two major methods to pick up the cut dimension: select a single cut dimension one at a time [5, 11, 14] or select multiple cut dimensions at a time [6, 9, 12]. When choosing a single cut dimension, the height of decision tree is usually higher than that by choosing more dimensions. But the node structure size is smaller because choosing multiple dimensions needs to keep more information. On the other hand, there are two major methods to separate the filters, some algorithms use prefixed as the filter separating method and thus create equal-sized subspaces for dividing the rule table. In other words, they only need to store the "cut bits" in decision tree's internal nodes instead of the keys (or cut-point).

The other method is to divide the rule table by using cutting endpoints. Each rule in the filters generates a range (or interval) between two endpoints. Only endpoints of ranges are used as cut-points. Choosing end-points has more flexibility than choosing prefix. Hicuts [5] and Hypercuts [9] both employ equal-sized cuts. They use a heuristic to decide how many cuts should be employed. The most important difference between Hicuts and Hypercuts is that Hicuts only cuts one dimension in an internal node but Hypercuts cuts multiple dimensions. Therefore, Hypercuts' tree depth is shorter than Hicuts.

Hypersplit [11] only cuts a single dimension in an internal node, but it employs end-point to find out the cut-point. First, for each interval, Hypersplit calculates the number of rules that cover the interval and store it in Sr[j] for $1 \leq j \leq M$, where M is the number of end-points. Then it chooses the smallest end-point m such that $\sum_{j=1}^{m} Sr[j] > \frac{1}{2}\sum_{j=1}^{M} Sr[j]$, which is called *heuristic weighted segment balanced strategy*. This strategy tries to make the sum of covering rules of all the intervals at the left side and right side of the end-point m equal. Hypersplit only separates subspaces into two parts. Furthermore, Hypersplit only picks up one dimension to cut, so the Hypersplit decision tree is a binary tree.

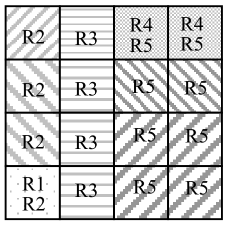
In this paper, we propose a packet classification algorithm that picks up multiple dimensions and cutting with end-point to make the height of decision tree much shorter. Then we propose a layered mechanism to reduce the memory consumption dramatically.

III. Proposed Algorithm

Our proposed algorithm focuses on two aspects. The first aspect is to pick up the dimensions and the second aspect is to decide the cut-point. We propose some heuristics for picking up dimensions and deciding cut- point.

Table 1. An example of a 2-D rule set.

Rule	Field-X	Field-Y
R1	[00,00]	[00,00]
R2	[00,00]	[00,11]
R3	[01,01]	[00,11]
R4	[10,11]	[11,11]
R5	[10,11]	[00,11]



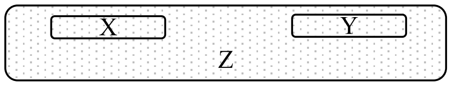


Figure 2. An example for rules distribution.

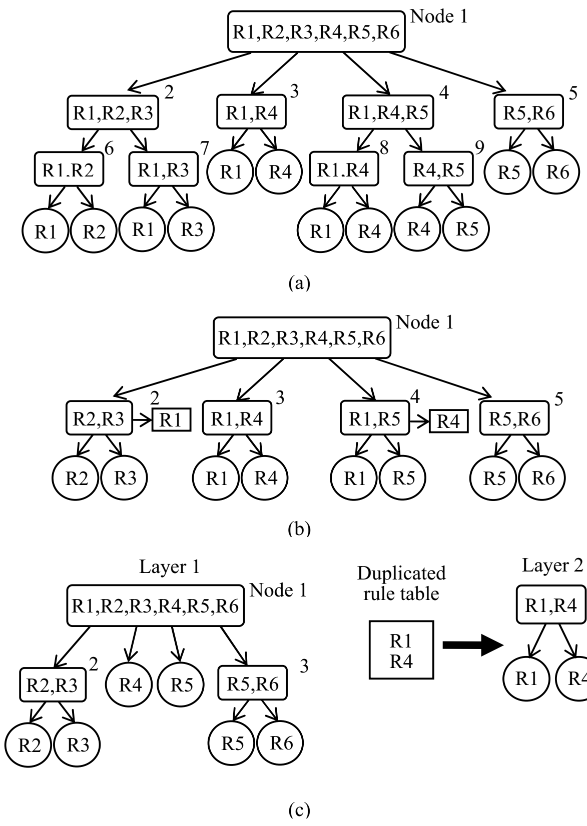


Figure 3. Pushing up heuristic and our optimization for rule duplication.

A. Select the cut dimensions:

1) Heuristic 1: Distinct field values. We consider the set of dimensions with larger number of distinct field values. For the example in table 1, the distinct field values in field-x are 3. R4 and R5 also have the same field-x value. 'Larger' dimension means the number of distinct field values for this dimension is greater than the mean number of distinct field values of all dimensions. For example if the five dimension's distinct field values of rules are 45, 15, 35, 10, 3 with mean 22, then we should choose dimensions first and third.

2) Heuristic 2: Average covering rules of all intervals. According to Hypersplit, the average number of covering rules of all intervals is calculated for each dimension. The dimension which has minimal average value is selected by Hypersplit. In our method, we choose those dimensions whose values are smaller than the average value of all dimensions.

3) Heuristic 3: The number of end-points. We choose those dimensions whose number of end-points is greater than average number of endpoints of all dimensions.

B. Space decomposition:

1) Heuristic 1 : Weighted segment-balanced. This is the heuristic weighted segment balanced strategy proposed by Hypersplit described above.

2) Heuristic 2 : 1/2 end-point. The cut-point *m* is selected such that the number of intervals at *m*'s left side is equal to that of *m*'s right side. That is, we choose the 1/2(lowbound end-point + upbound end-point) as the cut-point.

By combining three heuristics for selecting dimensions and two heuristics for selecting cut-point, there are six possibilities to divide the address space into subspaces. In our experimental results, for selecting the cut dimension, we choose distinct field values heuristic, and for select cut-point we choose weighted segment-balanced heuristic to obtain the best results of memory consumption and number of memory accesses.

Table 1 is a 2-D rule table. There are 5 rules and R1's priority is highest. Figure 1 (a), (b), (c), and (d) show the decision trees built by Hicuts, Hypercuts, Hypersplit and the proposed algorithm with the bucket size of 1.

In Figure 1(a), Hicuts employs the equal-sized subspace partition, and chooses only one dimension to cut for every internal node. Because only one dimension is selected at a time, the tree height of final decision tree is highest among all the schemes. Higher tree generates more internal nodes, and the memory storage become large.

In Figure 1(b), Hypercuts also employs the equal-sized subspace partition, but it chooses multiple dimensions at each internal node. So, the height of decision tree decreases dramatically. But, there is a critical drawback that some rules are duplicated many times. For example, R2 exists in 4 leaf nodes. It wastes lots of memory to store those duplicated rules.

In Figure 1(c), Hypersplit chooses cut-points. At the first level, the rules in each of three intervals at field-x are 2, 1, and 2 So, value 10 is used as the cut-point which divides the rule table into two groups, {R1, R2, R3} and {R4, R5}. At level 2, left internal node's rules in each of two intervals at field-x are 3 and 1. So, selecting 01 as cut-point can divide rules into {R1, R2} and {R3}. The right internal node's rules in each of three intervals at field-y are 1, 1, ad 2. So, choosing 11 as cut-point can divide rules into {R4} and {R5}. By this rules it could completes the decision tree. This cut-point selection algorithm of Hypersplit reduces the rule duplications effectively.

Figure 1(d) shows the resulting decision tree of our proposed algorithm. We combine the advantages from choosing multiple dimensions and an effective cut-point method. We can find out that cutting with end-point are more efficient (the duplicated rules are less), and the tree depth is effected by choosing only one dimension or multiple dimensions. So our algorithm could makes memory storage

Table 2. The information of rule tables.

	acl1 1K	acl1 5K	acl1 10K	ipc1 1K	ipc1 5K	ipc1 10K	fw1 1K	fw1 5K	fw1 10K
# of rules	916	4415	9603	938	4460	9037	791	4653	9311

Table 3. The detailed performance statistics of the proposed scheme with bucket size = 8.

		ACL1			IPC1			FW1		
		1K	5K	10K	1K	5K	10K	1K	5K	10K
Layer 1	Memory(KB)	17.95	102.79	149.72	15.43	133.47	367.95	27.95	276.70	594.08
	Avg height	6.01	7.68	10.15	6.48	8.19	9.28	5.97	9.23	10.14
	D-rules	13	197	686	93	800	1218	270	1340	2383
Layer 2	Memory(KB)	0.06	7.06	14.20	8.18	36.38	57.16	1.73	6.95	13.13
	Avg height	2.00	5.71	6.29	5.74	6.77	7.58	5.52	8.60	9.57
	D-rules	0	0	0	0	315	360	115	190	119
Layer 3	Memory(KB)	-	-	-	-	15.10	24.04	37.58	29.18	15.49
	Avg height	-	-	-	-	5.95	6.35	5.77	5.26	4.89
	D-rules	-	-	-	-	117	214	0	38	19
Layer 4	Memory(KB)	-	-	-	-	27.13	9.85	-	1.98	0.20
	Avg height	-	-	-	-	6.57	5.81	-	4.21	3.00
	D-rules	-	-	-	-	0	108	-	0	0
Layer 5	Memory(KB)	-	-	-	-	-	29.87	-	-	-
	Avg height	-	-	-	-	-	6.56	-	-	-
	D-rules	-	-	-	-	-	0	-	-	-
Total memory (KB)		18.01	109.86	163.92	23.62	212.10	488.89	67.27	312.85	622.71

Table 4. Four-layer construction for IPC1 10K table.

		IPC1
		10K
Layer 1	Memory(KB)	367.95
	Avg height	9.28
	D-rules	1218
Layer 2	Memory(KB)	57.16
	Avg height	7.58
	D-rules	360
Layer 3	Memory(KB)	24.04
	Avg height	6.35
	D-rules	214
Layer 4	Memory(KB)	210.15
	Avg height	7.27
	D-rules	0
Total memory (KB)		659.31

smaller and decreases the height of decision tree. Although our duplicated rules are more than Hypersplit, but we propose another improved architecture that can make the tree height lower and decrease the memory storage dramatically.

IV. Optimization

Rule duplication is a very serious problem in packet classification. It will cause a rule replicated many times and use a lot of memory to keep them. Due to rules distribution and cover property, we can not avoid this problem, and all we can do is to decrease this probability. Figure 2 shows an example. Rule X and rule Y are disjoint and rule Z covers rule X and rule Y. If we want to partition these three rules, no matter how the cutting operation is performed, the rule Z always needs to be replicated.

Figure 3(a) shows rule duplications in a decision tree. R1 exists in node 6 and node 7 and as a result, both the left child of node 6 and node 7 need to store R1. In the same way, R4 exists in node 8 and node 9, and both the right child of node 8 and left child node of 9 need to store the R4. This situation causes a lot of redundant rules. So, we must keep cutting the tree until the number of rules in the node is less than the bucket size. Rule duplication not only increases the memory storage but also increases the tree depth.

Hypercuts proposes a solution to tackle this problem, named "*Pushing Common Rule Subsets Upwards*". If all children have the same rules, then the parent node will create a rule list (i.e., bucket) to store this rules instead of duplicating them in its children. Figure 3(b) shows the solution by Hypercuts. R1 is stored in the rule list of node 2 and R4 is stored in the rule list of node 4. When traversing to node 2 and node 4, the rules lists belonging the internal nodes must also be searched.

Our algorithm tackles those duplicated rules by removing those duplicated rules, and uses them to create a duplicated rule table. Figure 3(c) shows how we decrease the tree depth and the number of node. In our algorithm, during constructing our decision tree, if we find a rule could be moved out, then when we traverse to another node which has the same rule, this rule should be eliminated. That ensures the rule not existing in this decision tree and eliminates the replication condition effectively. Then, according to our above heuristic, another decision tree is constructed from the duplicated rule table. When during search, all the decision trees have to be searched. Because we could implement the search on FPGA, we can traverse all the layered trees in parallel without increasing the search time.

Of course, pushing common rules subsets upwards also could be implemented on hardware and search the rules in internal nodes in parallel. But notice that, partial redundancy can't be pushed up which causes the rule still being duplicated many times. In Figure 3(a), nodes 2, 3, 4 have the same rule R1, but the node 5 doesn't have it. So, R1 can not be pushed up to node 1. Although R1 can be pulled up to node 2, but node 3 and node 4 also need to keep R1 in their child nodes. The pushing up heuristic can be regarded as local operation that the different sub-trees pushing operation is independent. So rule duplication condition still exists. Our experimental comparison

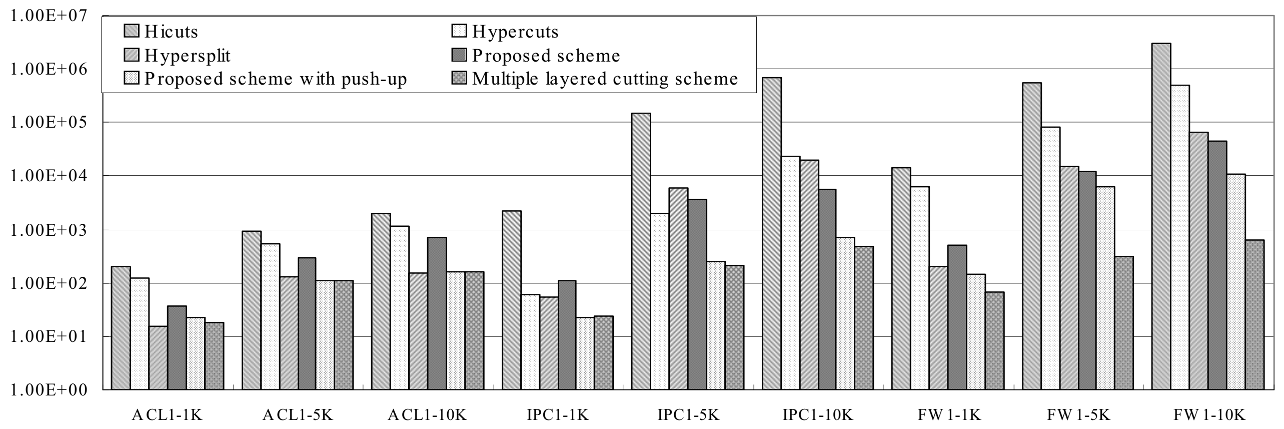


Figure 4. The memory usage (KB) comparison of Hicuts, Hypercuts, Hypersplit, proposed scheme (without optimization), proposed scheme with push-up and Multiple layered cutting scheme.

Table 5. Multiple layered cutting scheme memory reduction over Hypersplit.

	ACL1			IPC1			FW1		
	1K	5K	10K	1K	5K	10K	1K	5K	10K
Hypersplit (KB)	15	130	150	55	6000	20000	200	15000	66000
Multiple layered cutting scheme (KB)	18	109	163	23	212	488	67	312	622
Reduction	0.83	1.18	0.91	2.32	28.28	40.90	2.97	47.94	105.98

between pushing up and our layered scheme will be shown in section V.

Our data structure totally needs 112 bits for each internal node and leaf node. For internal node, 1 bit is needed to identify whether the node is an internal node or a leaf and 5 bits are needed to identify which cut dimensions are selected. We constrain the dimensions only up to 3 and so we need 80 bits to store the three cut-points, e.g., 32 bits, 32 bits, and 16 bits for two IP address fields and one port field. Also, we need 26 bits to store the address of leaf nodes. Because we can locate the sibling nodes to continuous address, so we only store the address for first child node and accesses the others by offset. For leaf nodes, the largest rule table we test is 10K, so we need 14 bits to discriminate rules, and the bucket size is 8, so we all need 14*8=112 bits for each leaf.

V. EXPERIMENTAL RESULTS

A. Data set

We use Access Control List (ACL), Firewall rules (FW) and IP Chains (IPC) with 1k, 5k and 10k, generated by the Classbench [8]. Table 2 is the number of rules for each rule table. And the bucket size for all leaf nodes is set to 8. The data structure of Hypersplit contains 64 bits (8 Byte).

B. Memory size and accesses

In Table 3, we present the memory size, average height of the decision tree and the number of duplicated rules in every decision tree. The number of trees is a trade-off between memory size and hardware cost. If we create more layers, then the total memory storage size would be smaller, but we need more hardware to implement search engines. Furthermore, according to the complexity and table size of rule table, the more complex is the rule table, the more parallel decision trees are needed. If we don't create more layers, the memory size would become much larger. For ACL1 and IPC_1k, two decision trees are enough to reduce the memory size significantly. But in IPC and FW, we need create more layers to reduce the total memory. Layer 1 is created by original rule table. When this tree is accomplished, i the duplicated rule table will be generated where the *D-rule* rows in Table 3 indicate the size of duplicated rule table. By using the duplicated rule table, we can create the decision tree of the next layer. All trees except the last one are implemented by using our proposed improvement heuristic. So, the last tree may need more memories and the tree depth is higher. Table 4 shows that in IPC1 10K rule table when only 4 layers are used. Although layer 4 only contains 214 rules, a total of 210.15 KB is needed. In the proposed layered cutting scheme, the duplicated rule table behind layer 2 is very small (not great than 360), but the rules in the duplicated rule set effect each other mutually and cause a serious duplication problem. Besides, in FW1 5k and 10K, the memory usage in tree 3 is lager than tree 2. The reason is that the rules moved from tree 2 to tree 3 may be heavily overlapped and thus, the rules remained in tree 2 are almost independent. As a result, the tree 2's memory becomes very small.

In Figure 4, we compare the memory consumptions for Hicuts, Hypercuts, Hypersplit, our proposed scheme without optimization, our proposed scheme with push-up and our layered cutting scheme. Hicuts and Hypersplit performance are according to the Hypersplit paper. Hicuts has the worst

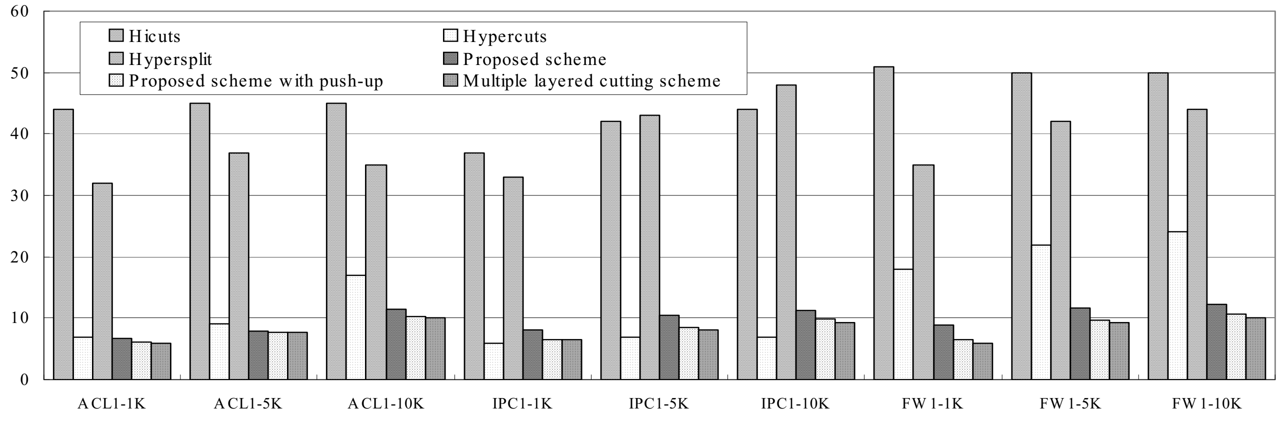


Figure 5. The memory access times comparison of Hicuts, Hypercuts, Hypersplit, proposed scheme (without optimization), proposed scheme with push-up and Multiple layered cutting scheme excluding leaf node sequential search.

Table 6. The memory access times comparison of proposed scheme (without optimization), proposed scheme with push-up and Multiple layered cutting scheme.

Tree depth	ACL1			IPC1			FW1		
	1K	5K	10K	1K	5K	10K	1K	5K	10K
proposed scheme	6.79	7.84	11.37	8.03	10.40	11.18	8.88	11.72	12.29
proposed scheme with push-up	6.02	7.73	10.23	6.59	8.50	9.88	6.57	9.70	10.68
multi-layered scheme	6.01	7.68	10.15	6.48	8.19	9.28	5.97	9.23	10.14

memory usage no matter which rule table is used due to its inefficient equal-sized and single dimension cutting. Hypercuts memory usage is better than Hicuts because the multiple dimensions are used for cutting. Our proposed scheme without optimization memory usage is larger than Hypersplit in small table or not complex rule table. But in bigger or more complex rule table such as IPC and FW tables of 5K and 10K rules, our proposed scheme is more efficient in dividing address space into subspaces. We also compare the optimizations between Push-up heuristic and our proposed layered cutting scheme. We could find out that our layered cutting scheme tackling rule duplication problem is much better than Push-up scheme significantly. Table 5 shows the comparison between our layered cutting scheme and Hypersplit as well as the factors by which our layered cutting scheme reduces Hypersplit's memory usage. In FW1 10K rule table, the reduction even gets up to 106 times, where memory reduction of the proposed scheme over Hypersplit is defined to be the ratio of the memories needed by Hypersplit and the proposed scheme.

In Figure 5, we compare the average numbers of memory accesses for Hicuts, Hypercuts, Hypersplit, our proposed scheme without optimization, our proposed scheme with push-up and our layered cutting scheme where the memory accesses for the sequential search in the leaf node are excluded. Hicuts and Hypersplit have larger numbers of memory accesses than Hypercuts and the proposed layered cutting scheme because they choose only single dimension to cut. Our layered cutting scheme and Hypercuts perform equally well for small rule tables. But, in larger rule tables, the proposed layered cutting scheme has better performance. In Table 6, we show the details between the one without optimization and the ones with optimizations. We can find out that optimization scheme not only improves the memory usage, but also the decision tree depth. We know that if the internal node's rule set size is great than the bucket size, this internal node can't convert to leaf node. Because our layered cutting scheme is able to reduce the duplicated rules, the rule set belonging to each internal node has more chance to convert to leaf node. So the height of decision tree becomes lower.

VI. CONCLUSION

We propose an efficient packet classification algorithm to reduce memory storage size and number of memory accesses. Many algorithms have good performance in small or not complex rule table, but in larger or complex rule table, their performance drops dramatically (or memory usage grows up irrationally). Our proposed algorithm could limit the memory growth efficiently. In other words, the performance improvement of the proposed schemes is not effected by the characteristics of rule tables. Our future work will be the FPGA implementation of the proposed scheme using pipelined architecture to increase the overall throughput.

REFERENCES

[1] F. Baboescu and G. Varghese. "Scalable Packet Classification," Proc. ACM SIGCOMM 2001, pp. 199–210, 2001.

[2] Yeim-Kuan Chang, "Efficient Multidimensional Packet Classification with Fast Updates," IEEE Transactions on Computers, VOL. 58, NO. 4, pp. 463-479, APRIL 2009 (SCI).

[3] Yeim-Kuan Chang, Y.-C. Lin, and C.-Y. Lin,"Grid of Segment Trees for Packet Classification,"The IEEE 24th International Conference on Advanced Information Networking and Applications (AINA-2010), 2010.

[4] E. Cohen and C. Lund. “Packet Classification in Large ISPs: Design and Evaluation of Decision Tree Classifiers”. Proc. SIGMETRICS 2005, pp. 73 – 84.

[5] P. Gupta and N. McKeown, “Packet Classification Using Hierarchical Intelligent Cuttings,” Proc. Hot Interconnects, 1999.

[6] P. Gupta and N. McKeown. “Packet Classification on Multiple Fields,” Proc. ACM SIGCOMM 1999, pp. 147–160, 1999.

[7] H. Lim and J. H. Mun, “High-Speed Packet Classification Using Binary Search on Length,” Proc. of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2007.

[8] D.E. Taylor and J.S. Turner, “ClassBench: a packet classification benchmark,” INFOCOM 2005, vol.3, pp. 2068-2079, 13-17 March 2005.

[9] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet Classification using Multidimensional Cutting.,” Proc. ACM SIGCOMM 2003, pp. 213-224.

[10] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvagel, “Fast and Scalable Layer Four Switching,” Proc. ACM SIGCOMM ’98, pp. 191-202, Aug. 1998.

[11] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li. “Packet Classification Algorithms: From Theory to Practice,” Proc. INFOCOM 2009, pp. 648–656, 2009.

[12] B. Vamanan, G. Voskuilen and T.N. Vijaykumar. “EffiCuts: Optimizing Packet Classification for Memory and Throughput,” Proc. ACM SIGCOMM 2010, pp. 207-218.

[13] T. Woo. “A modular approach to packet classification: Algorithms and results,” INFOCOM 2000, vol.3 ,pp. 1213-1222, 2000.

[14] B. Xu, D. Jiang and J. Li, “HSM: A Fast Packet Classification Algorithm,” Proc. of the 19th International Conference on Advanced Information Networking and Applications (AINA), 2005.

[15] B. Yang, X. Wang, Y. Xue and J. LI, “DBS: A Bit-level Heuristic Packet Classification Algorithm for High Speed Network,” Proc. 15th International Conference on Parallel and Distributed Systems, pp.260-267, 2009.

[16] B. Yang, et al., “Discrete Bit Selection: Towards a Bit-level Heuristic Framework for Multi-dimensional Packet Classification,” Proc. INFOCOM, IEEE, 2009.