2012 26th International Conference on Advanced Information Networking and Applications Workshops

CubeCuts: A Novel Cutting Scheme for Packet Classification

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

Yu-Hsiang Wang
Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, 701, Taiwan
p76991418@mail.ncku.edu.tw

Abstract*—Packet Classification is one of the most important services provided by Internet routers nowadays. Decision-tree based schemes, such as HiCuts and HyperCuts, are the most well-known algorithms. These schemes separate search space into many equal-sized sub-spaces. But both schemes have the same rule replication problem, which might cause large memory overhead. Although another decision-tree based solution, Hypersplits, was proposed to cut space according to end-points for reducing the memory requirement, we still observe that its rule replication problem doesn't be solved well and the memory requirement can still be improved. In this paper, we propose a scheme called CubeCuts to build a binary decision tree that does not generate any duplicated rule. By using the hybrid scheme that combines the CubeCuts and constrained HiCuts, we can have a memory-efficient data structure such that the entire rule table of up to10K rules can be fit into the on-chip memory of FPGA device. Our design is very suitable to be implemented with parallelism and pipeline. The experimental results show that the rule replication ratio is very low in all rule tables (ACL, FW, and IPC). The proposed parallel and pipelined architecture based on the hybrid scheme can achieve a throughput of 118 Gbps, which is enough to deal with the Internet traffic that is growing rapidly in recent years.

Keywords-FPGA; packet classification; decision-tree based algorithms; pipeline;

I. INTRODUCTION

Packet classification is a functionality of Internet routers that is needed for many important network services. It's usually mentioned by some features of networks like virtual private networks (VPNs), quality of service (QoS), network address translation (NAT), load balancing, traffic accumulating, differentiated services, etc. The basic packet classification problem is that router extracts the header fields of Internet packets and compares them with the rules pre-defined by network administrators. The extracted packet header fields consist of source address (SA), destination address (DA), source port (SP), destination port (DP), and protocol (Pro). Each rule is composed by these five fields and an action value. The action value of the matched rule will be taken by the packets, like "denied" or "accept" in Firewall (FW) rulesets. If one packet matches more than one rule, router must get the action value of the highest-priority rule.

Packet classification algorithms [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16] are still being improved by many researchers in recent years. It can be classified into many categories. The decision-tree based algorithm is one of most well-known solutions [5][11][12][16] such as HiCuts [5] and HyperCuts [12]. Both algorithms decompose the search space into many equal-sized sub-spaces, and it repeats recursively until the rules remaining the bucket associated with the node is no more than a threshold value. These algorithms have the same problems of high memory overhead when rule replications occur frequently. HyperSplit [11] proposed a cutting scheme based on endpoints to reduce the occurrence of rule replication. EffiCuts [16] proposed a heuristic for sub-spaces merging and a rule-partition skill to solve memory problem. Both HyperSplit and EffiCuts have improved decision-tree based algorithms by reducing the redundancy in wasting storage. We observe that the rule replication problem still exists and could be improved further. In this paper, we propose a novel cutting scheme called CubeCuts. CubeCuts builds a binary decision tree by selecting a subcube and dividing the search space into the one inside the subcube and the other outside of the subcube. The advantage is that CubeCuts does not replicate the rules. Since finding a prefect subcube to performance the search space decompose process is not easy, we propose a hybrid scheme that combines the CubeCuts and a constrained version of HiCuts to allow a small amount of replicated rules. The hybrid scheme provides a balance between the required memory usage and the height of the binary decision tree. Our experiments will show that the proposed CubeCuts outperforms HyperCuts and EffiCuts.

The rest of the paper is organized as follows. We describe the basic problem of existing decision-tree based algorithms in section II and propose a novel cutting scheme called CubeCuts in section III. Section IV shows our architecture design. The experimental results are given in section V. Finally, the conclusion remark is made in section VI.

II. RELATED WORK

A. Packet Classification

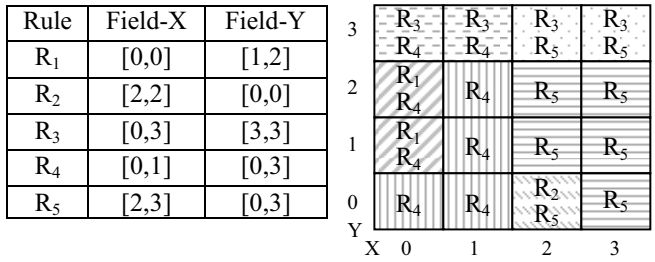
Packet classification can be solved by searching the ruleset (also called rule table) sequentially. If there are N rules in rule table, the searching time costs $O(N)$. The

978-0-7695-4652-0/12 $26.00 © 2012 IEEE
DOI 10.1109/WAINA.2012.110

decision-tree based schemes separate the rules into many small subsets, which are small enough to perform the linear search. The procedure of building decision tree is also called space decomposition, which means rules are divided according to the rule orientation. This procedure starts from root node, and rules are separated into child nodes recursively by space decomposition procedure until the number of rules is under a threshold value. Table I shows a sample rule table and its geometric view in two dimensional (2-D) space. The separated rules are stored in the corresponding leaf nodes also called buckets. If there are L leaf nodes, the average height of decision tree is $O(logL)$. So the searching time is reduced from $O(N)$ to $O(log\ L)$.

TABLE I. AN EXAMPLE OF 2-D RULE TABLE IN 2-BIT ADDRESS SPACE.

Rule	Field-X	Field-Y
R_1	[0,0]	[1,2]
R_2	[2,2]	[0,0]
R_3	[0,3]	[3,3]
R_4	[0,1]	[0,3]
R_5	[2,3]	[0,3]



B. Existing Algorithms.

1) HiCuts and HyperCuts

HiCuts [5] decomposes the searching space into many equal-sized subspaces recursively until the rules covered by each subspace is less than the pre-defined bucket size. The root node covers the whole searching space which contains all rules. HiCuts selects one dimension (or called field) to cut and decide how many subspaces should be taken for this dimension. Each child node covers one sub-space, and the parent node records the selected dimension and the number of subspaces (also called number of cuts) as *cut information*. Rules covered by root node are separated into each child node which covers them. If a rule cannot be covered by only one sub-space, a rule replication is needed. Figure 1 illustrates how HiCuts separates searching space, and the number of rules increases from 5 to 10 due to rule replication.

The searching process starts from the root node. Header of incoming packet is compared with the cut information of root to decide which child node should be visited next. When searching process reaches a leaf node, it searches the rules inside the bucket sequentially to get the matched rule.

HyperCuts [12] can be regarded as the multi-dimensional version of HiCuts. It can select more than one dimension to cut. In general, the searching time of HyperCuts is smaller than HiCuts because HyperCuts has a lower tree height. But the node size of HyperCuts is larger than that of HiCuts due to the larger cut information. HiCuts and HyperCuts face a major problem when the rule replication occurs frequently, which causes an unacceptable memory blowout problem for large rule tables.

2) HyperSplit

In Hypersplit [11], the size of separated sub-spaces is not equal. Hypersplit cuts the searching space by endpoints in a single dimension to improve the rule replication problem. Hypersplit calculates a value by heuristic called *weighted segment balanced strategy* to decide which endpoint should be selected. This heuristic tries to make the covering rules between left and right child nodes equal. Although the rule replication problem of Hypersplit is less than previous two algorithms, the node size of Hypersplit is larger than HiCuts and HyperCuts because of large cut information in endpoint format.

3) EffiCuts

EffiCuts [16] is proposed in two different aspects. One is node merging operation that tries to combine similar sibling nodes into one. After merging, the child nodes and the child pointer stored as cut information are reduced. The other one is that EffiCuts uses a grouping method to separate all rules before creating a decision tree. Each rule is classified and belongs to one group, depending on whether the value of each dimension is don't-care or not. Each group creates its own decision-tree independently. The grouping method works when the characteristic of rule table is significantly similar to Firewall and IPC. But EffiCuts has two node formats, it's very difficult to implement it in Hardware environment.

C. Analysis

1) Searching time

The key point of decision-tree based algorithm is how to construct a balanced decision tree with minimum tree height close to $O(log\ L)$, where L is the number of leaf node. Most of the decision-tree based algorithms were proposed by using heuristic to determine which dimension should be cut first and where should be cut better.

2) Memory requirement

Another factor we care about is the memory requirement. The memory storage depends on the node size and total number of nodes. Node size is related to what cut information is stored. The rule replication problem might consume more space decomposition procedures because the number of rules covered by current node is larger than predefined bucket size. The height of decision tree becomes deeper due to this phenomenon. It not only causes more nodes to be created but also increase searching time.

We focus on two things, *rule replication* and *balancing*. Especially, source port field and destination port field are represented in range format. The rule replication problem can't be avoided when any two ranges are partially overlapped or a rule has a don't-care value in these fields.

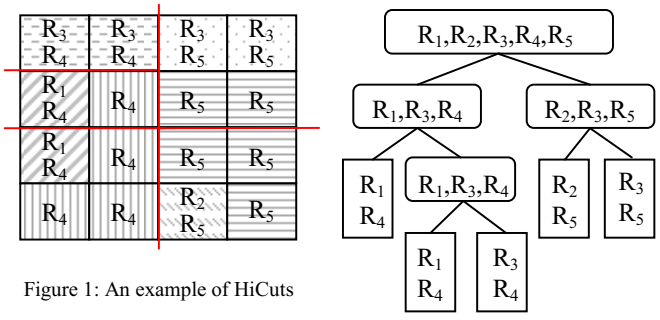


Figure 1: An example of HiCuts

The decision-tree based algorithms like HiCuts and HyperCuts deal them with large rule replication. When the size of rule table becomes larger, these two algorithms are hard to work in reality. Balancing is another factor we concern about. The number of rules in child nodes must be equal to each other to make tree balanced instead of skewing. A good decision-tree based algorithm must have a novel cutting strategy to reduce rule replication and keep decision tree balanced.

In this paper, we propose our scheme in two aspects. One is new grouping method, and another is space decomposing algorithm with low rule replication and balance factor. We implement our proposed scheme in Field-programmable Gate Array (FPGA). Hardware environment can design pipeline and parallel architecture to reach a high throughput, if the memory is small enough to be fit in on-chip memory of FPGA. However, the drawback is that the hardware logic designed to perform the search is more complicated than the existing schemes. In other words, when implemented in FPGA, more slices will be needed.

III. PROPOSED SCHEME

A. Definitions and Notations

Given two ranges in single dimension, $A = [a_1, a_2]$ and $B = [b_1, b_2]$, there are only three possible relationships between A and B.

Enclosure: $b_1 \le a_1 \le a_2 \le b_2$ or $a_1 \le b_1 \le b_2 \le a_2$.

Disjoint: $a_2 < b_1$ or $b_2 < a_1$.

Partially overlapped: A and B are neither enclosure nor disjoint.

In 5-dimensional space (also called **address space),** every rule has one range value in each dimension (prefix could be regarded as a range). If the range values of two rules R_1 and R_2 in any dimension are disjoint, R_1 and R_2 must be disjoint in the 5-dimensional address space**.** Let a rule be represented by ($[l_1, h_1]$, $[l_2, h_2]$, $[l_3, h_3]$, $[l_4, h_4]$, $[l_5, h_5]$). We define $L = (l_1, l_2, l_3, l_4, l_5)$ as the **low address** and $H = (h_1, h_2, h_3, h_4, h_5)$ as the **high address** of the rule and the rule locates in between the low and high addresses.

The proposed scheme includes two phases, the grouping phase and space decomposition phase. Based on many researches analyzing the general rule tables in the past, the rule replication problem is be lessened significantly by dividing rules into subgroups and construct the packet classification data structure for each subgroup independently. Then we can search each subgroup sequentially if it's a software based solution in uniprocessor environment or search all the subgroups in parallel if it's a hardware based solution implemented in the network processor, ASIC, or FPGA, etc. Therefore, like many other schemes, the first phase of ours proposed scheme is grouping the rules. In the second phase, we will apply a novel algorithm called CubeCuts to each subgroup. CubeCuts is proposed to be a scheme which does not generate any replicated rule. Zero replication cannot be done by any existing decision-tree based algorithm. As a result, the required memory will be less than the existing schemes. Also, the proposed will have higher search speed when the proposed CubeCuts scheme is implemented as a pipelined architecture in FPGA.

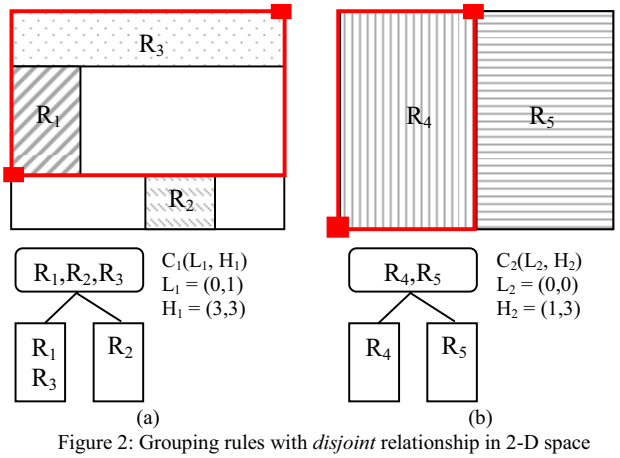


Figure 2: Grouping rules with *disjoint* relationship in 2-D space

B. Grouping Phase

A good grouping scheme helps to resolve the rule replication problem. The proposed grouping procedure consists of two steps. First, we group rules according to the distribution of don't-care values in all fields. For example, the values of source port field in ACL tables are all don't-care, the source IP and source port fields are usually don't-care in Firewall tables. There are $\sum_{i=0}^{5} C_i^5$ = 32subgroups, where C_i^5 is the number of subgroups in which all the rules have don't-care values in i fields. It is possible that some groups are empty for real-world rule tables. For instance, based on the analysis for various rule tables in [3], there are only 5-6 groups for ACL and FW tables of 10K rules. However, there exist 12 groups for IPC table of 10K rules. In order to keep the number of groups to a smaller size which is set to 3 in this paper, we pick up the two groups that contain the most and second most numbers of rules and merge all the remaining groups to be the third group. As a result, we have three groups. In the second step, the rules in each of the three groups are divided into more subgroups containing only disjoint rules based on the 5-dimensional disjoint relationship. In other words, after the second grouping step, all rules are 5-D disjoint in each subgroup. Figure 2 illustrates that the rule table in Table I is divided into two subgroups, {R_1, R_2, R_3} and {R_4, R_5}. R_1, R_2, R_3 are 2-D disjoint with each other, and so on between R_4 and R_5.

C. CubeCuts

The proposed space decomposition scheme is called *CubeCuts*. We decompose the address space by using the 5-dimensional sub-cubes. For a chosen subcube, the 5-D space is divided into two parts: one is the sub-space inside the subcube, and the other is the subspace outside the subcube. As done by the proposed CubeCuts, a binary decision is constructed. We don't consider the multiway decision tree because our focus is the pipelined architecture implemented in FPGA and thus the tree depth does not have a negative

```
// CubeCuts
01 Besti = Null;
02 Bestj = Null;
03 C_balance = ∞;
04 for i = 1 to N;
05   for j = 1 to N;
06       perform CubeCuts with subcube C(Li, Hj)
07       Diff = | |RL-Child| - |R R-Child| |
08       if ( (|RChild| == |RN| ) &&  (Diff < C_balance) )
09            C_balance = Diff
10            Besti = i;
11            Bestj = j;
12       endif
13   endfor
14 endfor
// Constrained Hicuts
15 Bestk = 0;
16 H_balance = ∞;
17 for k = 1 to 5;
18   Perform HiCuts on dimension k
19   Diff = | |RL-Child| - |R R-Child| |
20   if ( (|RChild| < |RN|*(1+ threshold %) )&& (Diff <H_balance) )
21      H_balance = Diff;
22      Bestk = k;
23   endif
24 endfor
// Selection Heuristic
25 if (H_balance < C_balance)  return HiCuts with dimension Bestk
26 else return CubeCuts with subcube C(LBesti, HBestj )
```

Figure 3: Selection heuristic between CubeCuts and CHiCuts.

impact on search speed. The child node which represents outside the subcube can be processed recursively without changing its cover range in 5-D address space. Another child node representing the address space inside the subcube can also be processed recursively in the reduced address space at the next iteration. Figure 2 illustrates the decision trees built for the two subgroups. R_1 and R_3 are inside subcube C_1((0,1), (3,3)) and R_2 is outside of C_1. Similarly, in figure 2(b), R_4 is inside subcube C_2((0,0), (1,3)) and R_5 is outside of C_2.

Since the rules in each subgroup are disjoint, it is easy to select a subcube to divide the rules into two parts such that no rule is replicated. The simplest subcube selection method in CubeCuts is to select the low and high addresses of one rule, L and H and cut the rules by the subcube C(L, H). We always can find subcube to decompose the address space without any rule replication. However, it is the worst case because there is only one rule inside the subcube C(L, H). As a result, the decision tree built this way is a skewed tree. However, computing the best subcube for cutting is a very time-comsuming task. Therefore, we use a simple heuristic to choose a subcube for cutting as follows. Let the number of rules in the current node to be processed be N. From the N lower addresses (L_1, …, L_N) and N high addresses (H_1, …, H_N), there are $N\times N$ pairs of (L_i, H_j) which can form a subcube for cutting, where i , j = 1 to N. Among the subcubes formed by all these $N\times N$ pairs of (L_i, H_j), we select the one such that balance factor called *C_Balance* is the minimum, where the balance factor is defined to be the difference between the number of rules inside and outside of the subcube. The detailed algorithm is shown in the first part of Figure 3.

Based on the observation from our experiments, sometimes allowing a small number of rule replications can avoid building a bad skewed tree of a very large depth. So, we decide to merge other existing decision trees to avoid this situation. Currently, we adopt HiCuts because the decision tree constructed by HiCuts is also binary and the nodes in HiCuts can share the same node format as CubeCuts. Since the rule replications in HiCuts can be numerous, we propose a modified version of HiCuts called *constrained HiCuts* as follows.

Constrained HiCuts (CHiCuts)- We constrain HiCuts by a rule replication ratio threshold in percentage. It means that the rule replication is less than *threshold* percent of the original number of rules after cutting. Specifically, no more than *threshold* rules are replicated every 100 rules after cutting. In the experiment, the threshold value is set to 30. The detailed algorithm is shown in the second part of Figure 3. Notice that the definition of the balance factor *H_Balance* for CHiCuts is similar to *C_Balance*.

D. Choosing between CubeCuts and Constrained HiCuts

We choose the cutting method between CubeCuts and constrained HiCuts. The more "balanced" one must be chosen. If rule replications are more than 30% in constrained HiCuts, constrained HiCuts cannot be selected. As shown in the last part of Figure 3, the actual cutting method selected between CubeCuts and CHiCuts is the one whose balance factor is smaller than the other. Although we use two methods in space decomposition procedure, we can still record the cut information in the same format. For the sample in Figure 2(b), the space is cut into two equal-sized subspaces. It can be regarded as a typical HiCuts, but we record L_2 (0,0) and H_2 (1,3), which is the same format as in the CubeCuts. It is easier to design the comparator circuit in hardware implementation.

When performing the search procedure, the header of incoming packet is compared with the cut information recursively until reaching a leaf node. For instance, in Figure 2(a), if the incoming packet header is K = (2, 2), then the left child node should be searched. If K = (3, 0), it will traverse the right child node because (3, 0) is not inside the subcube C((0,1), (3, 3)).

Our scheme reaches the goal of very low rule replication even in largest rule table. Table II shows the results of three different rulesets with sizes 1K, 5K, 10K. The left number is total rules after the decision tree is constructed, and the right

TABLE II. RULE REPLICATION RATIO

Ruleset size	ACL1	Firewall1	IPC1
10K	9897/9603 103%	12633/9311 136%	11533/9037 128%
5K	4823/4415 109%	6803/4653 146%	5300/4460 119%
1K	965/916 105%	1023/791 129%	1015/938 108%

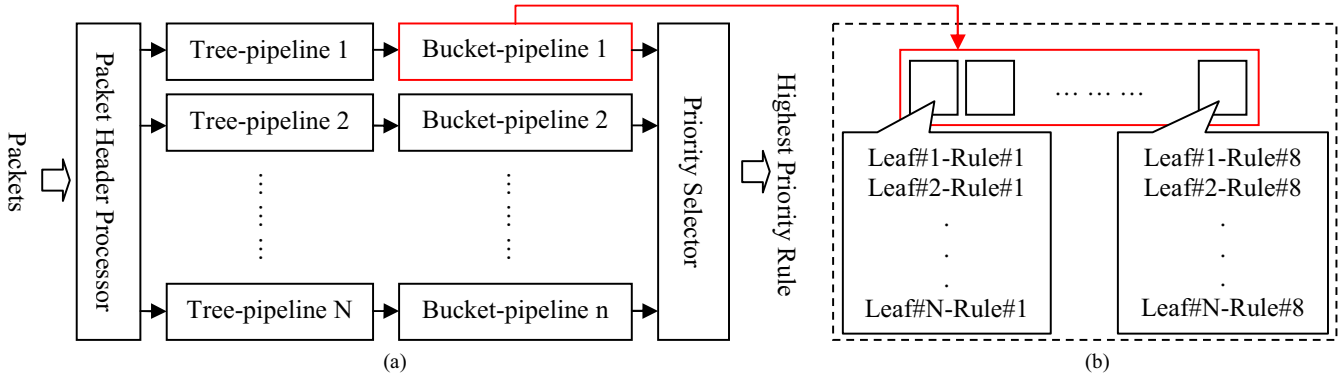


Figure 4: Search Engine with parallel and pipeline architecture.

one is the original size of rule table, and the rule replication ratio is shown as the percentage number below. Even for the worst case in Firewall1 of 5K rules, the rule replication ratio isn't larger than 150% (100% means no rule replication).

IV. Architecture Implementation on FPGA

Our proposed scheme is very suitable for hardware implementation. Either pipeline or the parallel architecture can improve throughput in hardware design. Each decision tree is mapped on one search engine, so we can search all decision trees simultaneously by the parallel architecture. The pipeline architecture is used in two parts, Tree-pipeline and bucket-pipeline. As we mentioned before, searching procedure traces the decision tree to the leaf node and search rules sequentially through this node (bucket). Cut information is stored in memory at corresponding stage in Tree-pipeline. Bucket-pipeline contains 8 pipeline stages following the tree-pipeline. All rules are stored in memory of bucket-pipeline with end-point format.

In our implementation, the protocol fields are not used to construct the decision trees. The protocol field is only used in the bucket search. Each node in Tree-pipeline needs 215bits, 1 bit set when it's a leaf node, 96*2 bits for two end-points (SA, DA, SP, DP), and 11*2 bits for two pointer of child nodes.

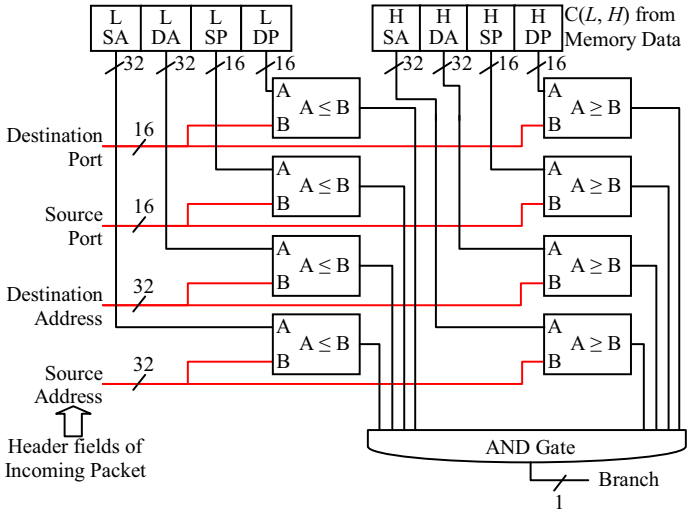


Figure 5: Architecture of branching circuit

Each rule stored in bucket-pipeline needs 222 bits including 104*2 bits for the complete information for a rule in the end-point format (SA, DA, SP, DP, Pro) and 14 bits for the number of rules for the largest rule table containing over 10,000 (10K) rules.

A. Dataset and Architecture Setup

We use Access Control List (ACL), Firewall rules (FW) and IP Chains (IPC) with sizes 1K, 5K and 10K, generated by the ClassBench [14] and select Xilinx vertex-5[17] xc5vfx200t device with speed grade -2 in our experimental results.

B. Memory Consumption

The number of total leaf nodes is very low because our scheme having low rule replication. The rule mapping method is shown in Figure 4. We compute block memory utilization by adding *number of entry* * *memory width* of each memory we use. Table III shows the memory storage requirement in hardware implementation.

Dual-ported Block-Rams on FPGA are used to implement necessary storage requirement. If the number of memory entries is less than 128, we use Distributed-Rams instead of Block-Rams by using LUT to reduce usage of the Block-Rams.

C. Circuit Design

Circuit design of a pipeline stage affects the maximum frequency our design can reach. If the logic operation is more complex, the gate delay between two registers must be longer. It makes throughput lower because of the long clock period.

The main factor of gate delay depends on the complexity of searching procedure. If the operation of visiting a tree node has many consecutive computations, it will cause a high gate delay. Our proposed algorithm needs to compare the values of four fields twice. It seems having a large amount of computation, but all computation is independent to each other. Figure 5 is the implementation of branching circuit. The branching circuit computes the child pointer to be taken. There are eight comparators in the branching circuit, two of which are 32 bits, and another two are 16 bits. The longest path of this circuit is a 32-bit comparator plus an

TABLE III. MEMORY REQUIREMENT (KBITS)

Ruleset size	ACL1	Firewall1	IPC1
10K	745.8	1026.4	967.9
5K	380.1	489.5	442.6
1K	77.9	85.4	83.9

TABLE IV. IMPLEMENTATION RESULT

	ACL1-10K	FW1-10K	IPC1-10K
# occupied Slices	11414/30720 37%	15505/30720 48%	24798/30720 80%
# Block Ram Utilization	195/456 43%	244/456 53%	194/456 42%
Clock period	5.276 (ns)	5.415 (ns)	5.398 (ns)

AND gate. So, our circuit takes short gate delay to reach very high throughput.

V. EXPERIMENTAL RESULTS

Our proposed scheme has two parts. The grouping result represents how many decision trees we should create. The tree-pipelines in parallelism are shown in Figure 4. There are 7 subgroups in ACL1-10K, 10 subgroups in FW1-10K, and 14 subgroups in IPC1-10K. The maximum heights of trees built for the three rule tables mentioned above are 22, 23, and 37, respectively.

Table IV shows the experimental results of hardware design in Xilinx ISE 12.2 environment, in terms of the slice resources and on-chip memory used.

Our design can achieve the throughput of over 118Gbps assuming the minimum packet size of 40 bytes. Table V shows the throughput comparison between our scheme and other methods [3][7][10][4][8][13][9]. SPMT by PW and PL [3] and Improved HyperCuts[7] have most similar conditions to our design. So, we show the details among these three methods in Table VI. We can see that our design has lowest total memory (1966Kbytes vs 612Kbytes vs 93Kbytes), Block-RAMs utilization (94% vs 89% vs 43%), and highest throughput (110.73 Gbps vs 80.23Gbps vs 118.46 Gbps).

TABLE V. COMPARISION WITH OTHER METHODS

Approaches	# of rules	Throughput (Gbps)
Our approach	9603	118.46
SPMT PW and PL[3]	9603	110.73
Improved HyperCuts [7]	9603	80.23
B2PC in ASIC [10]	3300	13.60
NTLMC [4]	12507	12.16
Power Saved HyperCuts [8]	≒25000	10.24
BV-TCAM [13]	222	10.00
2sBFCE [9]	4000	5.86

TABLE VI. COMPARISION WITH IMPROVED HYPERCUTS AND SPMT

	SPMT by PW and PL [3]	Improved HyperCuts [7]	Our Approach
Number of rules	9603	9603	9603
Total Memory(KB)	1966	612	93
Number of occupied Slices	6854 / 30720 24%	10307/30720 33%	11414/30720 37%
Number of Block-Rams utilization	429 / 456 94%	407/456 89%	195/456 43%
Frequency (MHz)	173.02	125.4	189.5
Throughput (Gbps)	110.73	80.23	118.46

VI. CONCLUSION

We proposed a novel decision-tree based algorithm called CubeCuts which is suitable in hardware environment. The grouping method with disjoint relationship greatly reduces the memory storage requirements. The size of on-chip memory is no longer a bottleneck no matter what kind the rule table is. In FPGA implementation, the decision making computation of searching procedure is a parallel and pipelined architecture. So we can reach the high throughput with low memory usage. Our future work includes building a multi-ways decision tree of CubeCuts without increasing clock period too much. If we reduce the number of pipeline stages (tree height), the hardware cost of FPGA could be further lowered.

REFERENCES

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

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

[3] Y.-K. Chang, Y.-S. Lin, and C.-C. Su, “A High-Speed and Memory Efficient Pipeline Architecture for Packet Classification”, Proc. IEEE FCCM, pp. 215-218, 2010.

[4] S. Dharmappurikar, H. Song, J. Turner, and J. Lockwood, “Fast Packet Classification Using Bloom Filters”, Proc. ACM/IEEE ANCS, 2006.

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

[6] P. Gupta , N. McKeown. “Algorithms for packet classification”, IEEE Network, Vol. 15, No. 2, pp:24–32, 2001.

[7] W. Jiang , V. K. Prasanna, “Large-Scale Wire-Speed Packet Classification on FPGAs”, Proc. ACM/SIGDA FPGA, 2009.

[8] A. Kennedy, X. Wang, Z. Liu, and B. Liu, “Low Power Architecture for High Speed Packet Classification”, Proc. ACM/IEEE ANCS, 2008.

[9] A. Nikitakis and I. Papaefstathiou, “A Memory-Efficient FPGA-Based Classification Engine”, Proc. IEEE FCCM, 2008.

[10] I. Papaefstathiou and V. Papaefstathiou, “Memory-Efficient 5D Packet Classification at 40 Gbps”, Proc. IEEE FCCM, 2008.

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

[13] H. Song and J. W. Lockwood, “Efficient Packet Classification for Network Intrusion Detection Using FPGA”, Proc. FPGA, pp. 238-245, 2005.

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

[15] D. E. Taylor. Survey and taxonomy of packet classification techniques. ACM Comput. Surv., Vol. 37, No. 3, pp:238–275, 2005.

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

[17] Xilinx, “Virtex-5 Family Overiew”, Product Specification, DS100 (v5.0), Feb. 6, 2009, at http://www.xilinx.com.