2009 15th International Conference on Parallel and Distributed Systems

DBS: A Bit-level Heuristic Packet Classification Algorithm for High Speed Network

Baohua Yang[*,†], Xiang Wang[ξ], Yibo Xue[†,‡] and Jun Li[†,‡]
[*]*Dept. Automation, Tsinghua University*
[†] *Research Institute of Information Technology, Tsinghua University*
[‡] *Tsinghua National Lab for Information Science and Technology, Beijing, China*
[ξ] *School of software engineering, University of Science and Technology of China, Hefei, China*
ybh07@mails.tsinghua.edu.cn, kojiroh@mail.ustc.edu.cn, {yiboxue, junl}@tsinghua.edu.cn

Abstract

Packet classification is one of the most critical techniques in many network devices such as Firewall, IDS and IPS, etc. In order to meet the performance requirement for high speed Internet (even higher than 10 Gbps), practical algorithms must keep better spatial and temporal performance. Moreover, as the size of rule set is increasing to tens of thousands, novel packet classification algorithms must have good scalability. In this paper, we propose a novel packet classification algorithm named DBS (Discrete Bit Selection) which takes a bit level heuristic design to partition the rule set effectively.

To the best of our knowledge, DBS is the first try to design a heuristic classification algorithm at bit-level. To evaluate the performance of our algorithm, DBS is deployed on a popular multi-core Network Processor platform, compared with two existing well-known algorithms. Experimental results show that DBS achieves 300% higher throughput than HiCuts and HSM, while the memory requirement is reduced to about 10% averagely. DBS works well especially with large rule set (10K), which trends a good scalability.

1. Introduction

Real-time packet classification is a foundational technique for a variety of Internet applications including traffic monitoring, Quality of Service (QoS) and security. The algorithm of packet classification has been widely used in network processing devices such as firewall, intrusion detection system (IDS) and intrusion prevention system (IPS), etc. For example, packet classification plays an important role in IDS[1, 2] to classify packets into different flows. Though the problem of packet classification has been studied for many years, researchers are still motivated to design more efficient algorithm due to the continual growth of network bandwidth and the increasing complexity of network applications. For example, 10 Gbps networks is required to transfer tens of millions 64-byte packets per second.

In the view of computational geometry[3], the problem of multi-field packet classification can be considered as a problem of point location in multi-dimensional space. It has been proven that the complexity bounds for n objects in k ($k > 2$) dimensions space are $O(\log n)$ in time while $O(n^k)$ in storage, or $O(\log^{k-1} n)$ in time while $O(n)$ in storage[4]. This theoretical performance is unacceptable in practice case, as many papers have pointed out. Researchers have been trying to design faster algorithms by process the rule set using heuristic methods[5, 6].

In this paper, we propose a novel approach that can perform well both on spatial and temporal performance (We have discussed a preliminary idea at[7] before). The algorithm is motivated by intuitive observation on the classification operations of some advanced heuristic algorithms, such as HiCuts. We introduce a more granular level heuristic design in which rule set will be partitioned more effectively. This method is amendable to be implemented at software platform, hardware platform or combined ones. The performance is evaluated on a real multi-core Network Processor (NP) platform in this paper.

Main contributions of this paper are:

1) A Bit-level Heuristic Packet Classification Algorithm

The algorithm proposed in this paper, adopts a bit-level heuristic to detect the inherent characteristics of the rule set. Due to the bit level heuristic, rule sets can be partitioned more efficiently, thus the storage required for data structures is significantly reduced. At the same time, to guarantee the speed of classification, two levels of flat structures are adopted, which require only two memory access times while searching.

2) Performance Evaluation on NP

To validate the performance of DBS on real NP platform, we built the classification application on the Cavium OCTEON 5860 network processor[8]; for comparison, we also implemented the traditional well-known algorithms: HiCuts[6] and HSM[9] on the same NP platform. Experimental results show that DBS

1521-9097/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPADS.2009.53

outperforms these existing best-known algorithms in terms of both spatial performance and temporal performance.

The remaining part of the paper is organized as follows:

Section 2 gives an introduction of related work; Section 3 illustrates the designing details of DBS while section 4 shows the experimental results of DBS, comparing with some well known algorithms; Conclusion and future works are discussed in section 5 and 6.

2. Related Work

Currently there are two major types of implementation of packet classification algorithms on commercial products: software-based implementation on general-purpose processor (such as IA CPU) and hardware-based ones on Application Specific Integrated Circuits (ASIC).

1) Software-based implementation

Although it is theoretically hard[4] to commonly deploy one single algorithm that works well for all cases, many effective algorithms were designed to improve performance in various practical cases, leveraging on the inherent characteristics of the real-life rule sets. Those algorithms fall into two categories: field-independent ones, such as RFC[10] and HSM[9], and field-dependent ones, including HiCuts[6] and HyperCuts[11]. Take the IPv4 protocol for example, the header of a packet contains 104 bits in 5 tuples, including two 32-bit fields (source and destination IP addresses), two 16-bit fields (source and destination ports), and one 8-bit filed (protocol type). These software-based algorithms could be categorized as below:

- Field-independent algorithms

Field-independent algorithms usually build their search structure in two steps: first use each of the 5 tuples data to build separate structures independently first, and then group them together. RFC and HSM perform independent parallel searches on indexed tables, and the results of the searches are combined into a final search result in several phases. Although these algorithms are fast in classification speed, they might require relatively large memory storage to store the search tables.

- Field-dependent algorithms

This type of algorithms treat different fields data dependently, thus usually grouping is not necessary at the final stage. HiCuts and HyperCuts both take intelligent and relatively simple decision-tree structures and heuristic methods to choose the field to cut. Since the field-dependent algorithms exploit more inherent relationship between different fields when building the decision tree, they typically achieve better tradeoff between time and space in practical cases. Although in most cases, these algorithms require less memory storage than field-independent search algorithms, however they cannot ensure a stable worst-case classification speed.

Although a lot of software-based algorithms have been proposed, most of them stagnate in theoretical analysis and simulation, without being widely implemented in commercial products.

2) Hardware-based implementation

Traditional network devices based on ASIC such as routers can achieve multi-Gbps processing speed, however, these devices are only limited to be used at the backbones[12], due to several issues:

- Programmability

Most of the ASIC architectures have special design for high performance which in turn leads to less general programming ability. This trends a tradeoff between performance and programmability.

- Special Chips Required

Special chips like Ternary CAMs can accelerate the packet processing speed. However, it requires too much power and board area to support large number of classification rules. Also, special chips usually mean higher cost, longer time-to-market and more difficulties in product upgrade.

Today, some researchers [12, 13] are seeking to combine the intelligence of software-based solution and the performance of hardware-based architectures, with the help of multi-core network processors (NP), which can support both flexible software programmability and powerful hardware-level packet processing.

3. DBS Algorithm

DBS is designed basing on two principles: (i) Use bit-level heuristics to split rules efficiently, and (ii) Adopt adaptive flat data structures to guarantee fast searching with low memory requirement. In this section, we will discuss the details of algorithm design. Table 1 shows a simple rule set example helping for discussion. Each rule only contains two tuples, while each tuple has two bits, so the whole length of one rule is four. Notice that although the example only shows prefix lookup, packet classification also needs range lookup for port tuples. However, a range can always be represented by one or several prefixes (see [13]).

Table 1 A simple example of rule set

Rule id	tuple 1	tuple 2
#1	00	00
#2	00	01
#3	01	00
#4	01	01
#5	**	**

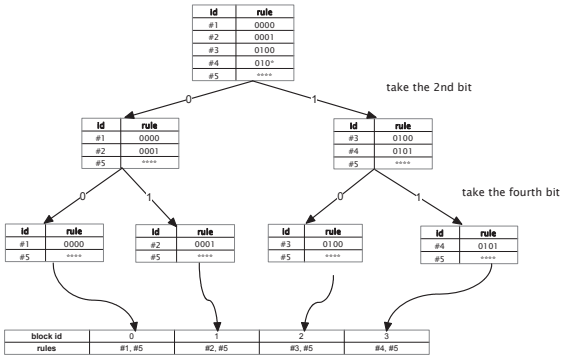

Figure 1 Preparation phase example of DBS

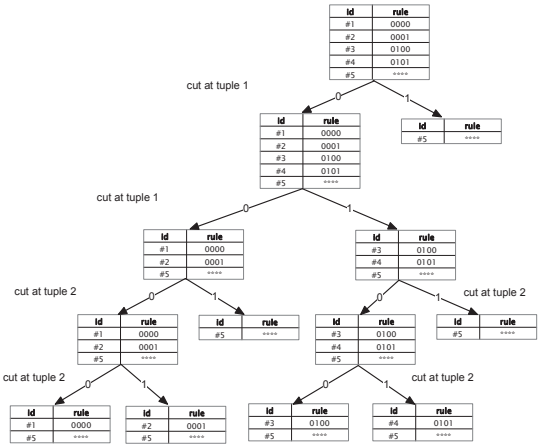

Figure 2 Example of HiCuts

3.1. Data structure

To describe the algorithm better, let's define some notations for the data structures used first:

E-bit

Consider the rule set in Table 1, each of the four bits at two tuples can split the rule set into two parts. For example, the first bit can split the rule set into 5 rules (which can get 0 at the first bit) and 1 rule (which can get 1 at the first bit). Other bits can also split rules into 3+3, 5+1 and 3+3. We can see that the second (or the fourth bit) minimizes the sub sets' size (the max size is 3), which means an effective partition. Such bit is named as "effective bit", or "*e-bit*" for short. Since e-bits will be the best ones to split rules apart; they can partition the rules effectively, even the size of rule set is quite large. Thus by checking several *e-bits* of a packet header, a large amount of rules that not match can be avoid during searching.

Mask vector

After selecting the *e-bits* of an L bits (for IPv4, $L = 104$) packet header **H**, a *mask vector* $\boldsymbol{V}$ with the same length can be defined as:

$\boldsymbol{V}_{[i]} = 1$, only if $\mathbf{H}_{[i]}$ is an *e-bit*;

$\boldsymbol{V}_{[i]} = 0$, otherwise.

where $\boldsymbol{V}_{[i]}$ means the i-th bit of **V**, and $\mathbf{H}_{[i]}$ means the i-th bit of **H**, $i = 1, 2, \ldots L$.

Notice the number of *e-bits* in the *mask vector* is always no larger than L, we define this number as l.

Also take the example in Table 1, if we select the second and the fourth bits as *e-bits*, then those two position of *mask vector* should be set to 1.Finally, we get the *mask vector* is: V = (0101), while the number of *e-bits* is two.

3.2. Preparation phase

The preparation phase includes two steps: the mask vector generation and the lookup table construction.

a) *Mask Vector* Generation

Suppose the number of rules in **R** is **N(R)**, with different V, **R** can be divided into different m subsets: S_1, S_2, …S_m ($m = 2^l$). To select *e-bits*, we define a function w as the weight function. Here with different weight functions, different heuristic partition methods will be taken.

In this paper, we define w as: $w = MAX(\mathbf{N}(S_i))$, Where $\mathbf{N}(S_i)$ is the size of subset S_i, because we try to minimize the subsets with the *mask vector*.

Our motivation is to select the most effective mask-vector which can split **R** into subsets as small as possible. Since we try to design the algorithm with good scalability, exhaustive selection is unpractical. Here two heuristic schemes are proposed. Though these two schemes are not the most optimized ones, they are simple and efficient for the selection. There are also some other novel algorithms for feature selection[14].

Sequential Forward Selection (SFS): This scheme starts with one bit and incrementally selects the others with checking w function value each step. First we select the most effective bit b_0 and partition the current rule set into two subsets. Then try to select another bit b_1 which trends the smallest w function value. Continue these first two steps until enough *e-bits* are selected. There are also other similar approaches such as sequential backward elimination(see [15]).

Heuristic Swap (HS): This scheme try to swap some bits of a given match vector to get smaller w function value. First we choose l *e-bits* from the 104 bits randomly and get the initial V_0, to divide R into 2^l subsets, and calculate out the w function value w_0. Then we try to swap each *e-bit* from V_0 with one from the unselected bits and get new w function value w_1. If $w_1 < w_0$, then accept the swapping, or else keep the bit and try the next one. To get better V, this scheme can be repeated after trying to swap all bits in the original V.

b) Lookup Table Construction

After the generation of v, the lookup table can be constructed with the following steps:

1) Set up a table T with length m ($m = 2^l$), where each cell $T_{[i]}$ stores a pointer to a storage block. Then for each rule r_i in **R**, do the next two steps;
2) Use V to mask out the e-bits and combine these bits as an index vector. Notice that the bit value of this index vector may be 0, 1 or * (* means both 0 and 1).
3) Fill r_i into the cells with the matched index.

In Figure 1, we give an example to use SFS to generate the *mask vector* for rules in Table 1, and then use *mask vector* to generate the lookup table. In this example, we take two steps to partition the rule set. Each step we only take one bit, at last we use only two bits to partition rule set into four sub ones, among which the largest one contains two rules.

Other field-level heuristic algorithms like HiCuts also can do the similar partition processing with a rules splitting trie structure, however, since the splitting is at field level, more bits are employed. Figure 2 shows the processing using HiCuts algorithm. We can see only with all four bits HiCuts can cut the rule set into sub ones no more than two rules.

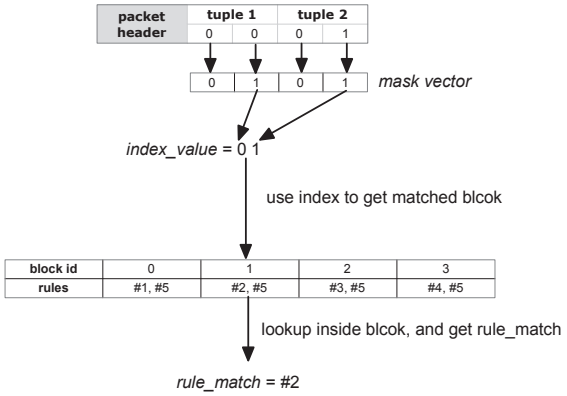

Figure 3 Classification example of DBS

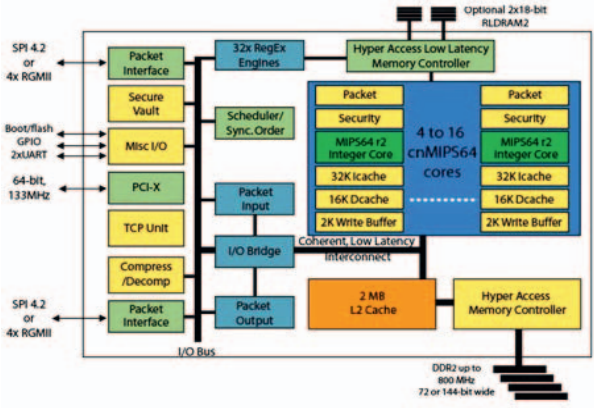

Figure 4 Block diagram of CN58XX

3.3. Classification Phase

With match vector V and the lookup table T, the classification can be finished in two steps:

1) Filter out l *e-bits* from each incoming packet's header with V, and combine them into a binary vector, take this vector's value as the lookup index i.
2) Index in the lookup table to get $T_{[i]}$, and check the rules in the block pointed by $T_{[i]}$ and find the best matched one.

The pseudo code for classification is as depicted as below:

```
00  int DBS_Classify(Vector mask_vector, Header hdr, Table lookup_table) {
01    /*Step 1: mask out the e-bits in the packet header, and calculate the index value*/
02    Vector index_vector = mask_vector & hdr;
03    int index = CalculateIndex(index_vector);
04    /*Step 2: index to blocks, and lookup_tablekup in blocks.*/
05
06    Block block = lookup_table[index];
07    int result = Lookup(block,hdr);
08    return result;
09    }
```

Figure 3 shows an simple example for the classification process of DBS. Since most blocks only contain few rules, the searching inside a block will be fast. Moreover, heuristic searching technologies can also be taken recursively inside.

As the rules in a block are stored continuously in memory space, network processors with cache like Cavium Octeon can take advantage. From this example, we can see that DBS only requires two memory access times to reach the rule block after the bit masking. The bit masking step is fast because this several bits operation can be done within cache, and special hardware can accelerate this step.

4. Experiments and Performance Analysis

To evaluate the performance of DBS objectively, we compare the algorithm's performance with HSM and HiCuts on four aspects: the spatial performance, the temporal performance, the scalability of the spatial performance and the throughput on NP.

To reach a good tradeoff between space and speed, we just set the number of *e-bits* to 16, which means we take 16 bits from the packet header for the partition. At the same time, because HiCuts uses linear searching inside leaf node to get a tradeoff between storage and speed, we

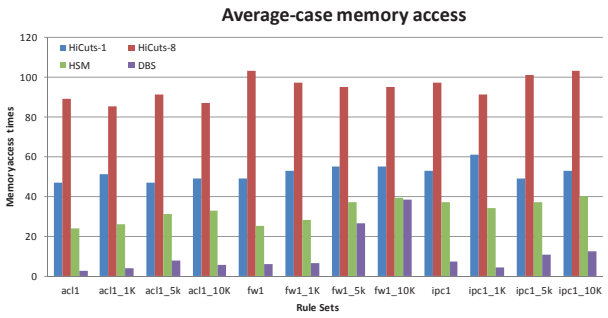

Figure 5 Average-case memory access times

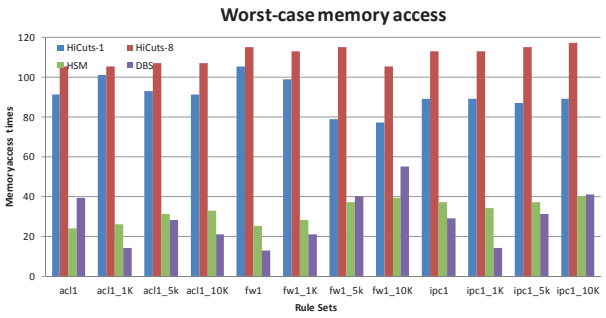

Figure 6 Worst-case memory access times

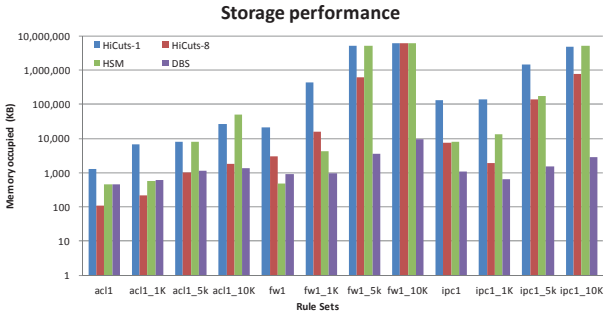

Figure 7 Memory storage comparison

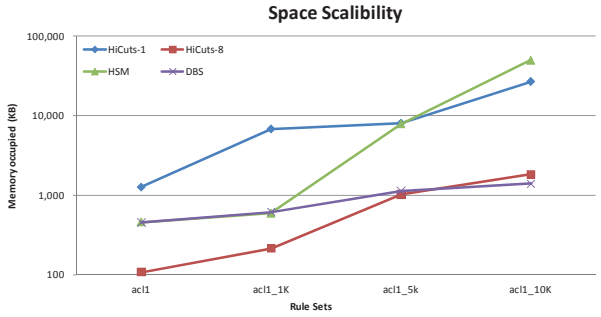

Figure 8 Scalability comparison

actually implement two versions of HiCuts with different sizes of leaf node: HiCuts-1 (only 1 rule in each leaf node) and HiCuts-8 (up to 8 rules in each leaf node). Thus, there are four implementations for three algorithms to compare: HiCuts-1, HiCuts-8, HSM, and DBS. All the algorithms are not only simulated by software, but also evaluated on a commercial NP platform.

4.1. Experiment Setup

4.1.1 Rule sets and traffic data

The experiments are carried out with publicly available real-life rule sets[16]. All the packet classification algorithms are evaluated on three different types of real-life rule sets, containing Access Control List (ACL), FireWall (FW) and IP Chain (IPC). We use 12 rule sets with different sizes from several hundreds to about ten thousand, e.g. the ACL1-10K rule set contains about 10,000 rules. All rules in the sets are of 5 tuples with 32-bit source/destination IP addresses, 16-bit source/destination port numbers, and 8-bit transport layer protocol. More details about the rule sets can be found at[16].

The traffic for experiments is generated by Spirent SmartBits 600B performance analysis system[17], which supports up to 4 SmartMetrics Gigabit Ethernet ports. Since there are 3 Gigabit Ethemet ports at the frontend of the NP testbed, the full throughput of experiments are limited up to 3 Gbps.

4.1.2. Test bed

The software simulation results are obtained on a PC (2.0GHz dual-core, 4GB DDRII memory), while all programs are C-code and complied with -O2 option under Ubuntu 8.04 Linux system.

The performance of throughput is evaluated on the Cavium OCTEON 5860 multi-core platform.

Figure 4 shows the block diagram[18] of CN58XX series multi-core platform. Each Cavium OCTEON CN5860 processor contains 16 cnMIPS64 running at 750 MHz; the memory include 4 GB DDR2 SDRAM with 2MB L2 cache shared among all cores. There are two programming modes provided by Cavium SDK: Linux mode (with a Linux OS) and Simple Executive mode (without OS supporting). We program codes at the Simple Executive mode for fast packet processing.

4.2. Performance Results

4.2.1. Memory Access Times

Both average-case and worst-case memory access times are illustrated in Figure 5 and Figure 6. As Figure 5 shows that DBS has the least average-case access times, in most rule sets, the average-case access times is only

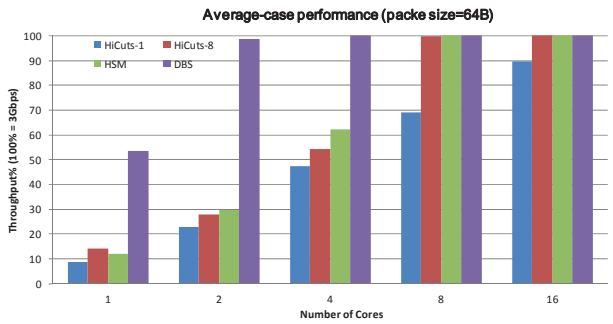

Figure 9 Average-case throughput performance (rules:ACL1-10K, #cores:1~16)

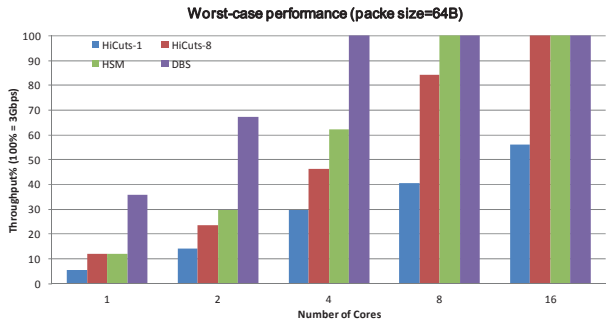

Figure 10 Worst-case throughput performance (rules:ACL1-10K, #cores:1~16)

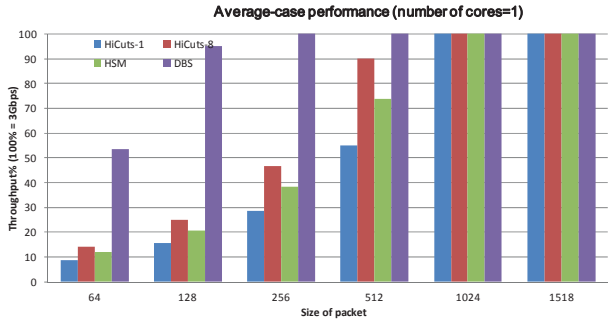

Figure 11 Average-case throughput performance (rules:ACL1-10K, packet: 64 ~1518 B)

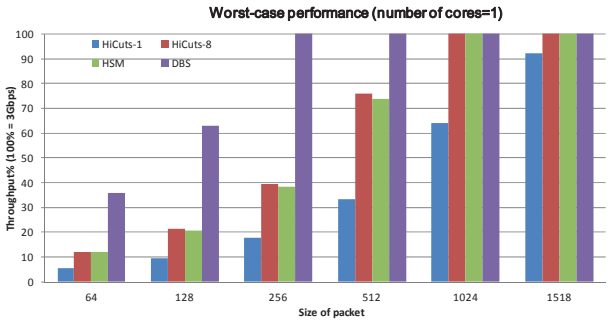

Figure 12 Worst-case throughput performance (rules:ACL1-10K, packet: 64 ~1518 B)

about 5% ~ 10% of that of HiCuts (both HiCuts-1 and HiCuts-8), while 10% ~ 20% of that of HSM. From Figure 6, we can see that the worst-case access time of DBS is only about 30% of that of HiCuts (both HiCuts-1 and HiCuts-8), while is nearly the same with that of HSM.

We notice that HiCuts-8 needs more memory access times than HiCuts-1 in average case. This is because the linear searching inside leaf nodes increases the access times. Though DBS also adopts linear searching inside blocks, however, the size of blocks can be keep small because of the effective partition.

4.2.2. Memory Storage

In Figure 7, we can see that the memory used by DBS is at least an order of magnitude less than that of HSM and HiCuts for most of the rule sets, especially for large ones (such as ACL1-10K, FW1-10K and IPC1-10K). Take rule set ACL1-10K for example, both HSM and HiCuts-1 require about 25 ~ 40 MB memory while DBS only requires about 1 MB. Though with rule set acl1 and acl_1K, HiCuts-8 can obtain better space performance than DBS, however, it requires more than DBS in all other rule sets, especially with large or complex ones.

We also notice that the memory required by all the algorithms on FW rule sets is larger than ACL and IPC ones. This indicates that FW rule sets have more complicated inherent structures. For the same size, rule sets of FW will generate more complicated sub-spaces in the multi-dimensional classification space than ACL and IPC rule sets. We think FW rule sets can validate the robustness of classification algorithm. With the largest FW rule set (FW1-10K), only DBS can finish the processing with requiring less than 10 MB memory, while both HSM and HiCuts (HiCuts-1 and HiCuts-8) are failed to finish building for too much storage required (larger than 4 GB).

4.2.3. Scalability of Spatial Performance

We also evaluated the scalability of storage requirement for all the three algorithms. We believe this is another important evaluation factor for packet classification algorithms because the size of classification rule set is growing larger quickly while large and cheap low latency access memory units (SRAM, etc.) are still not easy to be designed. Thus if one algorithm has a weak scalability, it will not work well with larger and more complex rule sets in the future, even it can perform well at current rule sets.

In Figure 8, we give the result of the spatial performance with different sizes of the ACL rule sets, which shows that DBS performs a better scalability than the other three algorithms. With rule set growing 10 times

larger (from ACL1-1K to ACL1-10K), the memory occupied by DBS only doubles, while that by other algorithms become about ten times more.

4.2.4. Throughput Performance

Figure 9~Figure 12 show the evaluation results of throughput performance on Cavium OCTEON 5860 multi-core platform. The rule set is ACL1-10K, which is the largest one that HiCuts and HSM can process successfully.

Figure 9 and Figure 10 show the average-case and worst-case throughput performance with different number of cores while the packet size is 64 Bytes. For the minimum 64 Bytes Ethernet packet, only DBS can achieve nearly 100% throughput (3Gbps) with 2 cores in average, while HiCuts-8 and HSM need 8 cores, and HiCuts-1 cannot achieve the full throughput even with all the 16 cores. Even in the worst-case, DBS is the only one which reach 100% throughput with 4 cores.

Figure 11 and Figure 12 show the throughput results on only 1 MIPS core with different packet sizes (64 Bytes ~ 1518Bytes). When input packet is 128 bytes, DBS can reach nearly 100% throughput in average-case and higher than 60% (about 2 Gbps) in worst-case. This result is about 300% higher than all other three algorithms.

4.3. Discussion

Theoretically, DBS has the same worst-case memory access time with HiCuts; however, with granular heuristics, DBS can gain better performance with real-life rule sets. At the same time, more effective partition can reduce the storage requirement. This indicated that rule set based decomposition algorithms may obtain more advantage than space decomposition ones.

When comparing the spatial performance, we notice that for small and simple rule sets (acl1 and acl_1K), DBS requires more storage than HiCuts-8, while for large or complicated rule sets, DBS works better. There is several reasons: first the structure of trie uses less internal nodes when leaf node is large for small rule set; secondly, DBS use a fixed 2^{16} = 65,536 size lookup tables, this means a large number of redundancies of the default rule. While for large and complicated rule sets, trie structure will need more steps to cut, which need more internal nodes.

5. Conclusion and Future Work

In this paper, we propose a new bit-level heuristic algorithm for multi-dimensional packet classification called Discrete Bit Selection (DBS). Unlike other existing solutions, this algorithm takes advantage of a heuristic partition of rule set at bit level, which allows it to better explore the inherent characteristics in rule set and split rules more effectively. In order to evaluate the performance of the proposed algorithm, we implement DBS on a real NP platform, along with some other well-known algorithms such as HSM and HiCuts. Experimental results on Cavium OCTEON NP platform show that DBS achieves superior temporal and spatial performance to the existing algorithm in most cases, especially when testing with large rule sets, which are expected to be the trend of the next generation Internet. Specially, DBS also achieves better scalability with different sizes of rule sets, which indicate it can work robustly with even larger rule sets in future.

Although the experimental results are encouraging, current work is still preliminary. In this paper, we set a determinate number of *e-bits*; however heuristics can be utilized to get proper number of *e-bits* with different environments (rule sets, platform, etc.). This would dig out more potential of DBS, and make DBS more practical and robust with more complicated and more compressive classification problems. Another interesting problem is the generation schemes of *mask vector*. Though there are many feature selection approaches, a special light-weight one will be more reliable. And we plan to evaluation DBS with more rule sets in more types of platforms.

6. Acknowledgements

This work has been supported by the National High-Tech R&D Program (863 Program) of China under grant No.2007AA01Z468. The authors would like to thanks Yaxuan Qi, Fei He, Lianghong Xu and other colleagues in the Network Security Lab[19] of Tsinghua University, for their suggestions.

7. References

[1] M. Roesch, “Snort, the de facto standard for intrusion detection/prevention,” 2006; http://www.snort.org.

[2] V. Paxson, “Bro: A system for detecting network intruders in real-time,” *Comput. Networks*, vol. 31, no. 23, 1999, pp. 2435-2463.

[3] M. De Berg, et al., *Computational geometry: algorithms and applications*, Springer, 2008.

[4] M. Overmars and A. van der Stappen, “Range searching and point location among fat objects,” *Journal of Algorithms*, vol. 21, no. 3, 1996, pp. 629-656.

[5] T.Y.C. Woo, “ Modular approach to packet classification: Algorithms and results,” *Proc. IEEE INFOCOM*, IEEE, 2000, pp. 1213-1222.

[6] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," 1999, pp. 34-41.

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

[8] "OCTEON™ Plus CN58XX Multi-Core MIPS64 Based SoC Processors," 2009; http://www.caviumnetworks.com/OCTEON-Plus_CN58XX.html.

[9] B. Xu, et al., "HSM: A fast packet classification algorithm," *Proc. International Conference on Advanced Information Networking and Applications*, Institute of Electrical and Electronics Engineers Inc., 2005, pp. 987-992.

[10] P. Gupta and N. McKeown, "Packet classification on multiple fields," *Computer Communication Review*, vol. 29, no. 4, 1999, pp. 147-160.

[11] S. Singh, et al., "Packet Classification Using Multidimensional Cutting," Association for Computing Machinery, 2003, pp. 213-224.

[12] T. Sherwood, et al., "A pipelined memory architecture for high throughput network processors," 2003, pp. 288-299.

[13] V. Srinivasan, et al., "Fast and scalable layer four switching," ACM New York, NY, USA, 1998, pp. 191-202.

[14] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, 2005, pp. 491-502.

[15] Y. Yang and J. Pedersen, "A comparative study on feature selection in text categorization," MORGAN KAUFMANN PUBLISHERS, INC., 1997, pp. 412-420.

[16] D.E. Taylor and J.S. Turner, "ClassBench: A packet classification benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, 2007, pp. 499-511.

[17] Spirent, "Spirent SmartBits SMB-600 2-slot Portable Chassis," 2009; http://www.smartechconsulting.com/SMB-600-Spirent-Smartbits-SMB600-2-Slot-Portable-Chassis?sc=11&category=595.

[18] "OCTEON™ Plus CN58XX Multi-Core MIPS64 Based SoC Processors," 2009; http://www.caviumnetworks.com/OCTEON-Plus_CN58XX.html.

[19] "Network Security Lab," 2009; http://security.riit.tsinghua.edu.cn/wiki/NSLab.