Practical Multituple Packet Classification Using Dynamic Discrete Bit Selection

Baohua Yang, *Member*, *IEEE*, Jeffrey Fong, *Member*, *IEEE*,
Weirong Jiang, *Member*, *IEEE*, Yibo Xue, *Member*, *IEEE*, and Jun Li, *Member*, *IEEE*

Abstract—Multituple packet classification is one of the key technologies, and often the performance bottleneck in modern network devices. Devices such as firewalls demand fast packet classification on very complicated rule sets of large size, which is still challenging today. This paper proposes a practical packet classification algorithm named dynamic discrete bit selection (D^2BS), which achieves high classification speed while requiring low storage. D^2BS employs dynamic heuristic schemes at bit level, to explore the inherent characteristics of the rule sets. D^2BS has been implemented on various platforms including Intel-architecture, multicore network processor, and FPGA, and is compared with the state-of-the-art solutions. Experimental results on real-life rule sets show that the memory storage required by D^2BS is at least one to two orders of magnitude lower than that of the existing work, while the speed is much higher. With 64-byte Ethernet packet and 10K size ACL rule set, D^2BS achieves a throughput over 10 Gbps on Cavium OCTEON CN5860 multicore network processor and over 135 Gbps on Xilinx Virtex-5 FPGA, which outperforms the existing work under the same test environment. All results promise that D^2BS is a highly practical solution to satisfy vigorous requirements.

Index Terms—Packet classification, quality of service, high-performance network

1 Introduction

With the rapid development of traffic engineering technologies on Internet, multituple packet classification has been considered as one of the foundational techniques of network devices in both industry and academia. Network devices, such as firewall, and Intrusion Detection/Prevention System (IDS/IPS), require a high-performance real-time processing. For example, devices deployed in current 100-Gbps networks are expected to classify hundreds of millions packets per second, and the operations in modern financial exchanger must ensure the latency within 100 microseconds between the exchanger and data centers [1]. These requirements will become stricter due to the continual growth of network bandwidth and the increasing complexity of network applications.

On the other hand, fast accessing memory (e.g., TCAM) is still relatively small and expensive [2], which limits the storage requirement of the algorithm. All these requirements make it difficult to design a practical and general algorithm that is suitable to be deployed on various platforms. Thus, although the problem of packet classification has been studied for many years, researchers are still motivated to design more efficient and practical solutions.

- *B. Yang and J. Fong are with the Department of Automation, Tsinghua University, Beijing 10084, China. E-mail: ybh07@mails.tsinghua.edu.cn, jfong111@gmail.com.*
- *W. Jiang is with Juniper Networks, Inc., Sunnyvale, CA 94089. E-mail: weirongj@acm.org.*
- *Y. Xue and J. Li are with the Research Institute of Information Technology, Tsinghua University, Beijing, China. E-mail: {yiboxue, junl}@mail.tsinghua.edu.cn.*

Manuscript received 8 Oct. 2011; revised 7 May 2012; accepted 25 July 2012; published online 1 Aug. 2012.
Recommended for acceptance by Y.-D. Lin.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-10-0712.
Digital Object Identifier no. 10.1109/TC.2012.191.

The main task of multituple packet classification is to lookup a matched rule r in a given N-tuple rule set R with the highest priority for an incoming packet, where "match" typically means that five tuples (source and destination address, source and destination port, protocol) in the packet header H should be covered by r. From computational geometry's view, this can also be considered as a point location problem in multidimensional space. It has been proven that the theoretical complexity bounds for classifying N nonoverlapping regions in $K(K > 2)$ dimensional space are $O(logN)$ in time while $O(N^K)$ in storage, or $O(log^{K-1}N)$ in time while $O(N)$ in storage [3]. Since the typical size of today's rule sets has reached tens of thousands and is still increasing [4], practical packet classification algorithm should also be designed with good scalability to the size of rule sets, especially the scalability in spatial performance.

In this paper, we propose a novel packet classification algorithm named Dynamic Discrete Bit Selection (D^2BS), which performs well both on spatial and temporal performance. The algorithm is motivated by inherent observation on the real-life rule sets. We design dynamic heuristics at fine-grained level and combine different types of data structures to optimize the performance. To evaluate the performance, we implement D^2BS on various platforms including Intel architecture-based (IA-based) platform, Cavium OCTEON CN5860 multicore network processor (NP) [5], and Xilinx Virtex-5 FPGA [6]. The evaluation provides encouraging results over different sizes and types of real-life rule sets. Our previous work DBS [7] has also tried a preliminary idea of classifying packets at bit level, while D^2BS improves it with more effective dynamic heuristics and provides more sufficient evaluation results on various types of platforms.

The primary contributions of this paper over existing work are as follows:

- *Performance*. D^2BS achieves high performance on various types of platforms, including IA-based, NP, and FPGA. Taking the Xilinx Vertex-5 FPGA for example, on 10K size rule set with 64-byte packets, D^2BS achieves over 135-Gbps throughput, while the latency is only several ns. This is the best result as far as we know. High performance on various platforms proves D^2BS a practical algorithm to be deployed in the production quality network devices.
- *Scalability*. With evaluations results on rule sets of different sizes and types, D^2BS shows a good scalability. For example, with ACL rule sets whose sizes increase from 1K to 10K, memory requirement of D^2BS is less than double. The Memory Per Rule (MPR) of D^2BS also holds very stable for all rule sets. This good scalability promises that D^2BS will perform well on more complicated and larger rule set in future.
- *Flexibility*. To achieve good flexibility among different types of rule sets, D^2BS utilizes a *P-Function* to adjust its data structures' parameters intelligently. By choosing different *P-Functions*, D^2BS can be utilized for various performance requirements. For example, when deploying D^2BS on different types of rule sets, D^2BS takes different *P-Functions* to optimize the performance.

The remainder of this paper is organized as follows: Section 2 summarizes the related work. Section 3 details the design of the D^2BS algorithm and Section 4 describes the implementation and analyzes the experimental results. In the last section, we draw a conclusion of our work.

2 Related Work

Currently, there are two major types of packet classification algorithms according to their designing motivations in academic community and industry: searching space partition ones and rule set partition ones.

Searching space partition algorithms try to partition the searching space into smaller subspaces to reduce the lookup complexity, such as RFC [8] and HSM [9]. Space partition algorithms usually take advantage of the indexed table data structures based on different tuples to obtain high searching speed. RFC and HSM both perform independent parallel searches on their indexed tables, and the results of the searches are cross-producted into a final search result in several phases. Although this type of algorithms is fast in classification speed, they might require relatively large memory storage to store the searching tables, which degrades their performance sharply for the large rule sets.

Based on the idea of decision tree, many effective rule set partition algorithms were designed to improve the performance in practical cases, leveraging on the inherent characteristics of the real-life rule sets. These algorithms mostly aim to cut the large rule set into smaller ones, including HiCuts [10] and HyperCuts [11]. HiCuts and HyperCuts both take heuristic methods and relatively simple decision-tree structures to select the tuple to cut at. Since these 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. In most cases, rule set partition algorithms require less memory storage than space partition ones; however, they cannot ensure a stable worst case classification performance.

Although a lot of novel algorithms have been proposed, most of them stagnate in theoretical analysis or only *simulate* on commodity platforms (mostly on general-purpose processor), without being widely implemented in commercial products. The main reasons can be categorized twofold:

- *Performance limitation.* Today's devices like routers require more than tens of Gbps processing performance, which is hardly achieved by those algorithms on commodity platforms. On the other hand, commercial network devices require a stability of performance, which demands a performance guarantee in worst case. To the best of our knowledge, none of current algorithms can meet those two requirements at the same time.
- *Scalability shortage.* Some of current algorithm can work well for small rule sets, such as RFC and HiCuts. However, the storage and speed performance will fall when rule set sizes increase. A practical algorithm should provide reasonable scalability to the sizes of rule sets.

For those reasons, traditional devices mainly utilize algorithms based on special hardware, such as Application Specific Integrated Circuits (ASIC) chips. These ASIC-based network devices 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 programmability. This trends a tradeoff between performance and programmability.
- *Scalability.* 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 are seeking to combine the intelligence of software-based solution and the performance of hardware-based architectures, with the help of multicore network processors and FPGA, which can support both flexible software programmability and powerful hardware-level packet processing.

Qi et al. [13] implemented an intelligent trie-based algorithm named HyperSplits on Cavium OCTEON CN3860 NP which achieve good performance in both time and space. On FPGA platform, Jiang and Prasanna [14] proposed two optimization methods for the HyperCuts algorithm to reduce memory consumption, which can achieve a throughput of 80 Gbps. To the best of our knowledge, the work of Qi et al. [15] is the first one that gives results over 100 Gbps throughput on Xilinx Vertex-5 platform. All these works show an appealing way to design novel packet classification algorithm on platforms like NP and FPGA.

TABLE 1
A Simplified Rule Set Example

Rule id	Tuple 1	Tuple 2
$\#r_0$	00	1*
$\#r_1$	01	00
$\#r_2$	10	*0
$\#r_3$	11	11
$\#r_4$	11	1*
$\#r_5$	1*	00
$\#r_6$	**	**

3 Dynamic Discrete Bit Selection Algorithm

D^2BS is designed based on two principles: 1) utilize dynamic heuristics to split the rule set efficiently, and 2) combine different data structures to optimize both the time (classification speed) and space (memory storage) performance. In this section, we first introduce the definitions of *E-Bit*, *M-Vector D-Table*, and *S-Block*, after that we explain the algorithm. An example rule set is utilized to help introduce the idea, as shown in Table 1 and Fig. 1.

3.1 Terminology

3.1.1 E-Bit

Suppose r is a rule of the given rule set R, and r is constructed by numbers of bits that belong to different tuples. The values of these bits can be "0", "1", or "*". We observe that some bits will partition R more "effectively" (In D^2BS, we judge this effectiveness by an heuristic function, *J-Function* which is given in Section 3.2.1) than others. We name these "effectively partition" bits as "*E-Bits*". D^2BS will employ these *E-Bits* to partition R into smaller subrule sets. Notice that the subrule sets may be disjoint or not, and here we utilize the term "partition" for both cases.

3.1.2 M-Vector

Suppose the header of the incoming packet is H (L bits), and we have generated n *E-Bits*. By them, we can filter out n bits from H ($n \leq L$) at the corresponding position with *E-Bits*. To facilitate this filtering, we design a masking vector as the M-Vector, whose definition is described as follows.

Definition 3.1 (M-Vector). *An M-Vector V is a bit vector that satisfies: $V[i] = 1$ only if bit_i is an E-Bit; otherwise, $V[i] = 0$.*

Also taking Table 1, for example, if we select bit 2 and bit 4 as the *E-Bits*, then the corresponding *M-Vector* is $V = (0101)$, while the number of the *E-Bits* is 2.

3.1.3 D-Table

With the *M-Vector* V, we can filter n bits from the packet header H, where $V[i] = 1 (0 \leq i < n)$. These bits are combined as a bit-string whose possible values range from 0 to $2^n - 1$. To achieve a fast indexing from these values, we design a dynamic indexing table T as the "*D-Table*". For n bits, T consists of 2^n cells, where each cell stores a pointer, respectively. Notice the bit-string with "*" may result in several values, which means 1-bit string may match different cells.

3.1.4 S-Block

S-Blocks are memory blocks that are pointed by T's cell. Each *S-Block* stores a subset of the rule set R. All rules in *S-Blocks* are stored orderly from high to low by their priority.

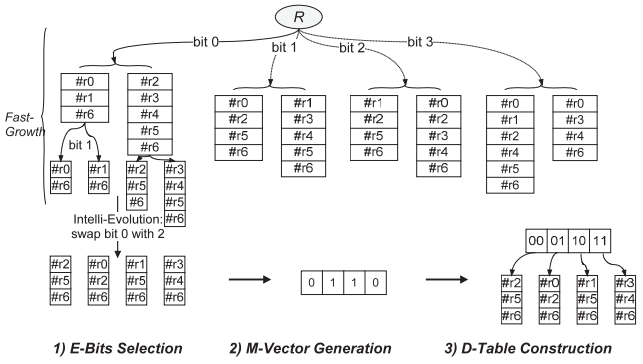


Fig. 1. The preparation phase on the example rule set.

3.2 Preparation Phase

With definitions given, the preparation phase is described in three steps: *E-Bits selection*, *M-Vector generation*, and *D-Table construction*.

To describe the algorithm clearly, a simplified example of rule set is given in Table 1, and the preparation phase on the rule set is shown in Fig. 1. The rule set consists of seven rules, and each rule only contains $N = 2$ tuples, while each tuple has 2 bits, so the length of each rule $L = 4$. Notice that although the example only shows prefix lookup, real packet classification also needs range lookups. However, a range can always be represented by one or several prefixes [16].

3.2.1 E-Bits Selection

In this step, there are two problems to address: one is that which bits are *E-Bits* (*E-Bits* choosing) and the other is how many *E-Bits* should be chosen (length optimization). We solve these two problems by designing two dynamic heuristic functions: the Judging Function (*J-Function*) and the Performance Function (*P-Function*). Although heuristics cannot always guarantee the optimization, we take advantage of these two heuristic functions to achieve good balance between the calculation cost and the optimization by well designs.

J-Function. J-Function is used to judge which bits are *E-Bits*; hence, the efficiency of *J-Function* will affect the performance of classification directly. Suppose the rule set R is partitioned into m subsets ($R_0 \ldots R_{m-1}$) by an *E-Bits* set e, *J-Function* can be given based on the optimal goals of the partition. Suppose the optimal goal is to minimize the maximum size of all subsets, then *J-Function* is defined as follows:

$$J(R, e) = -\max_{i=0}^{m-1}(NumRule(R_i)), \quad (1)$$

where $NumRule(R_i)$ means the number of rules in R_i. Hence, with maximizing $J(R, e)$, we can choose *E-Bits* which are the best ones to partitioned the global rule set R into small subsets effectively.

Notice with different optimal goals, we can define various *J-Function*. For example, with the optimal goal of minimizing the size of subsearching space, *J-Function* is defined as follows:

$$J(R, e) = -\max_{i=0}^{m-1}(NumSubspace(R_i)), \quad (2)$$

where $NumSubSpace(R_i)$ means number of subspaces generated by R_i. Hence, with different requirements, D^2BS can flexibly adjust its performance to satisfy them dynamically.

P-Function. *P-Function* is used to decide the number of *E-Bits*, i.e., to optimize the length of n. Choosing smaller n will result in larger *S-Blocks* which means a longer searching time inside; on the contrary, choosing large n will result in small *S-Blocks* but a large *M-Vector*, which may occupy too much memory storage. So an optimization of n can gain a better tradeoff between the temporal and spatial performance. Suppose we choose n based on the memory storage, then the *P-Function* can be defined as follows:

$$P(R,n) = Mem(V) + \sum_{i=0}^{2^n-1} Mem(R_i), \tag{3}$$

where $Mem(V)$ means the memory used by the *M-Vector*. Notice that we can also define different *P-Functions* according to various performance requirements such as the worst case classification time.

3.2.2 M-Vector Generation

M-Vector V is built with the processes of *E-Bits* choosing and length optimization. Here our motivation is to select the most effective V which can split R into subsets as small as possible. Since D^2BS is designed to provide good scalability, intuitive selection methods such as exhaustive searching will be impractical. Here, we design a heuristic-based scheme named as "Intelli-Swap" (*I-Swap*), which includes two steps: *Fast-Growth* and *Intelli-Evolution*.

For the *Fast-Growth* step, we generate V by fast selection methods. Several method can be taken, for example, Sequential Forward Selection (SFS) [17] which increases V's length by testing each bit with the *J-Function* and the *P-Function*. This step generates a local-optimal V which is prepared for the *Intelli-Evolution* step. The length of V is calculated by the heuristic functions.

After that, for the *Intelli-Evolution* step, we try to swap some selected *E-Bits* with the unselected ones, and examine if the new combination is better in optimization. A limitation number can be set to restrain the time of this process. As shown in Fig. 1, the *E-bits* by the *Fast-Growth* are bit 0 and bit 1. After the *Intelli-Evolution* step, bit 0 is swaped with bit 2. Finally, the *E-Bits* results are bit 1 and bit 2.

3.2.3 D-Table Construction

After the generation of V, the *D-Table* T can be constructed by the following steps:

1. Set up T with length 2^n, where each cell $T[i]$ stores a pointer to an empty *S-Block*. Then pick each rule from R orderly from higher to lower priority, for example, r.
2. Use V to mask r at the *E-Bits* positions, which results in a bit-string i of length n (where n is the number of *E-Bits*).
3. Insert r into the bottom of the *S-Block* pointed by $T[i]$.

Notice that for one rule, it may generate bit-string of several different values. Here, we duplicate and store the rule into different *S-Blocks*. This may increase the storage of D^2BS but facilitates the processing of the classification phase. On the other hand, we pick each rule following the priority order and only insert the new rule into the bottom of the *S-Block*; thus, the rules stored in the *S-Blocks* are also ordered. Actually, most real-life rule sets are generated with priority already [4]. Even when the rule set is not in order, it's easy to sort them before the construction.

Consider the rule set in Table 1, for example. Suppose 2 *E-Bits* are selected from all the 4 bits. Then the preparation phase can be shown as Fig. 1. In this example, each of the four bits can split the rule set into two parts. For example, the first bit can split the rule set into two subsets: three rules of {#r_0, #r_1, #r_6} (which covers 0 at the first bit) and five rules of {#r_2, #r_3, #r_4, #r_5, #r_6} (which covers 1 at the first bit). The other bits (bit 2-4) can also split rules into subsets as {#r_0, #r_2, #r_5, #r_6} + {#r_1, #r_3, #r_4, #r_5, #r_6}, {#r_1, #r_2, #r_5, #r_6} + {#r_0, #r_2, #r_3, #r_4, #r_6}, and {#r_0, #r_1, #r_2, #r_4, #r_5, #r_6} + {#r_0, #r_3, #r_4, #r_6}. We take the partition with the smallest subsets' size as the optimization goal in the Fast-Growth step. After the Fast-Growth step, we choose bit {0, 1} as the possible *E-Bits*, and the resulted maximal size of the subset is 4. Then with the Intelli-Evolution, we test to swap bit 0 with bit 2, and successfully get the new *E-Bits* as bit {1, 2}. The new result induces the max subset's size from 4 to 3, which is an optimized result.

After the *E-Bits* Selection, the *M-Vector* and the *D-Table* are constructed. The entire procedure of preparation is shown in Fig. 1.

At last, with *E-Bits selection*, *M-Vector generation*, and *D-Table construction*, all data structures for classification are built. As a summary, the pseudocode of the preparation phase is demonstrated in Algorithm 1. Notice the preparation phase is usually processed offline in advance, and the generated data structures can be loaded into the system during maintenance.

Algorithm 1. Algorithm of the Preparation Phase.
Input:
R
Output:
V, T
1: $V = \{0\}, J' = -\infty$//Init
2: //Fast-Growth
3: **repeat**
4: **for** $i \in \{i : V[i] = 0\}$ **do**
5: $J = J(R, e)$
6: **if** $J' < J$ **then**
7: $s \leftarrow i$
8: $J' \leftarrow J$
9: **end if**
10: **end for**
11: V[s]=1
12: n++
13: **until** $P(R,n) \geq P_{upper}$
14: **for** $i < LIM_{EVO}$[1] **do**
15: TryEvolution(V)//Intelli-Evolution
16: **end for**
17: //D-Table Construction

1. LIM_{EVO} is the limitation constant of evolution time or number of tries, which can be preset by users.

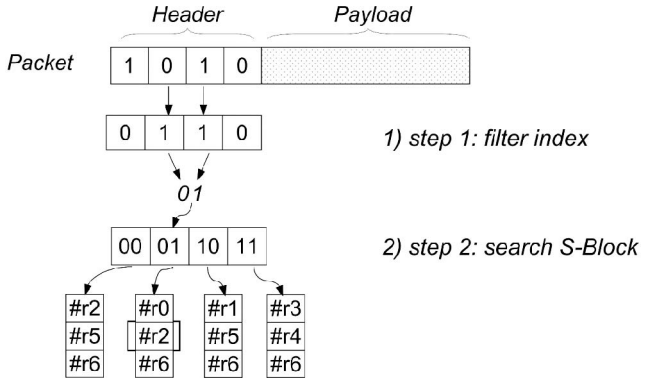


Fig. 2. The classification phase on the example rule set.

18: **for** $r \in R$ **do**
19: $bstring = JoinBit(r\&V)$
20: **for** $i \in bstring$ **do**
21: $T[i]$.append(r)
22: **end for**
23: **end for**
24: **return** V, T

3.3 Classification Phase

After the preparation phase, we now get the data structures of V and T. With them, the classification phase can be performed in two steps, as illustrated in Algorithm 2.

1. For each incoming packet header H, filter out all bits with V, at the *E-Bits* positions, and combine them into a bit string *bstring*. Calculate *bstring*'s value as the index i, which will be utilize in the second step.
2. Check the rules in the *S-Block* that is pointed by $T[i]$, and then find out the best matched one. Typically, "best matched" means the rule that is both matched and with the highest priority. Since the rules in the *S-Block* are already ordered during the *D-Table* construction (see Section 3.2.3), the process is simplified to find out the first matched one.

Algorithm 2. Algorithm of the Classification Phase.
Input:
V, T, H
Output:
r// The result rule.
1: $bstring = JoinBit(H\&V)$
2: $i = getValue(bstring)$
3: **for** $r \in T[i]$ **do**
4: **if** $isMatched(H, r)$ **then**
5: goto END
6: **end if**
7: **end for**
8: **return** r

Fig. 2 shows the classification process of D^2BS on the example rule set in Table 1. Since most blocks only contain a few rules, the searching inside a block will be fast even with linear lookup. Certainly, other searching technologies can also be taken recursively inside.

On the other hand, the rules in a block are stored continuously in the memory space, hence network processors, for example, Cavium OCTEON can take advantage of their cache mechanisms. From this example, we can see that after the bit masking, D^2BS only requires one memory access times (find the correct cell of the *D-Table* with the index) to reach the small rule block. The bit masking step is also fast because these bit operations can be done within cache, and special hardware can also accelerate this step.

3.4 Complexity Analysis

Consider rule set R consists of N rules and n *E-Bits* are chosen. If D^2BS is utilized as space-sensitive, or the *P-Function* is defined based on the memory storage, less *E-Bits* will be chosen to reduce the memory storage cost. On the other hand, if D^2BS is utilized as time-sensitive, or the *P-Function* is defined based on the memory access times, more *E-Bits* will be chosen to cut down the memory access times.

Here, we give out the theoretical complexity of D^2BS. The time complexity is $O((\frac{2}{3})^n \cdot N)$, while the storage complexity is $O((\frac{4}{3})^n \cdot N)$. Thus, with different *E-Bits* number, we can easily tune the performance of D^2BS on various platforms. Furthermore, in the practical cases, the complexity will be even better. As indicated in [4], for both source and destination address prefixes, the prefix nesting thresholds. A more detailed analysis of the complexity is given as the following.

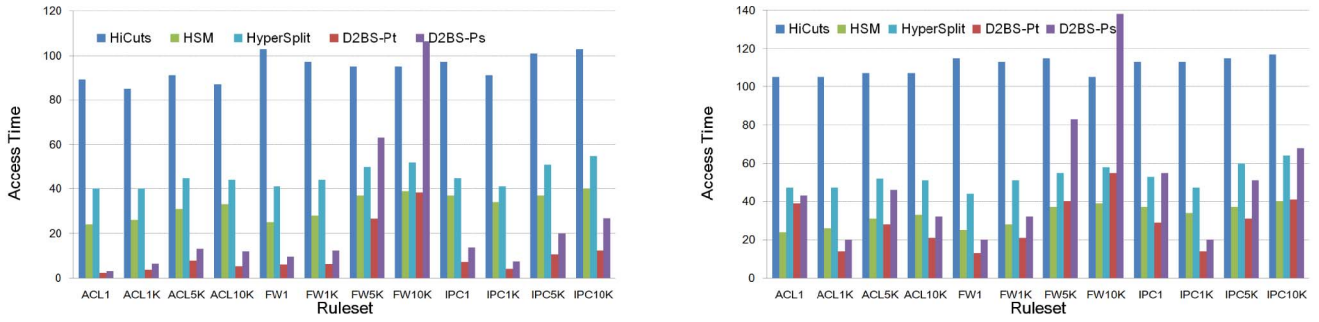
Proof. Suppose rule set R consists of N rules, and n *E-Bits* are selected. Thus, the length of the *D-Table* is 2^n. Suppose the length of the largest *S-Blocks* is L. It is clear that the time complexity is $O(L)$, while the storage complexity is $O(2^n \cdot L)$.

The next step is to find out the complexity of L. Each bit may have three types of values: "0", "1", and "*". With an *E-Bit*, the rule set will be partitioned into two subsets. Suppose the size ratio between the subsets and the original rule set is ρ. Then the size of the partitioned subsets will be $\rho \cdot N$. Thus, with n *E-Bits*, $O(L) = O(\rho^n \cdot N)$.

In theory, on average (Assume each type of bit values has the same probability, i.e., $\frac{1}{3}$.), $\rho = \frac{2}{3}$, as the rules with "*" will be copied into both subsets. For example, if we have a rule set with three rules: $\{r_1, r_2, r_3\}$, and the first bit of each rule is "0", "1", and "*". We choose the first bit as the *E-Bit*, this will partition the rule set into two subsets: $\{r_1, r_3\}$ and $\{r_2, r_3\}$. However, in the practical case, $\rho < \frac{2}{3}$, as the "*" bit has low existence probability in real-life rule sets. Taking the ACL series rule sets, for example, the ratios of the "*" bit are only about 3.0 and 6.4 percent on the source and the destination tuples, respectively. Thus, in the worst case, $O(L) = O(\rho^n \cdot N) = O((\frac{2}{3})^n \cdot N)$.

In summary, the theoretical time complexity is $O((\frac{2}{3})^n \cdot N)$, while the storage complexity is $O((\frac{4}{3})^n \cdot N)$, where n is the number of *E-Bits*, and N is the size of the rule set R. □

Notice that in the practical cases, the complexity is better than this theoretical bound. As indicated in [4], the prefix nesting thresholds for both source and destination address prefixes. It is also clear that with larger n, better temporal performance will be achieved with more storage cost. Specially, when $n = lgN$, then we have constant (relative to N) time of $O(N^{1+lg\frac{2}{3}})$, and space of $O(N^{1+lg\frac{4}{3}})$.



(a) Memory access time comparison of the average case.

(b) Memory access time comparison of the worst case.

Fig. 3. Memory access time comparison of the average and the worst case between D^2BS and other well-known algorithms.

4 Implementation and Evaluation

To evaluate D^2BS's performance objectively, we first implement D^2BS on our IA-based platform with both time-sensitive *P-Function* (D^2BS-Pt) and space-sensitive *P-Function* (D^2BS-Ps). Performance results are given in term of time, space, and scalability, compared with other well-known algorithms such as HiCuts, HSM, and HyperSplit. To evaluate the real throughput performance of D^2BS on the industrial platform, we also test D^2BS on Cavium OCTEON CN5860 multicore NP (16 cores with 750 MHz, and the memory is 4-GB DDRII) and Xilinx Vertex-5 FPGA platform (99.64-146 MHz with 2-MB BRAM). Notice that there is only 2-MB L2 cache on our NP platform, and the memory storage of the FPGA platform is also limited.

All the experiments are carried out with publicly available real-life rule sets, containing the types of Access Control List (ACL), FireWall (FW), and IP Chain (IPC). We use 12 rule sets of all types for evaluation, with different sizes from several hundreds to about 10,000, for example, the ACL10K rule set contains about 10,000 rules. All rules are of five 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 in [4].

4.1 IA-Based Implementation

4.1.1 Memory Access of Average Case

The memory access of the average case is one of the important evaluation parameters, which is usually used to evaluate the algorithm's temporal performance in practice. We compare D^2BS with other well-known algorithms. As Fig. 3a shown, both D^2BS-Pt and D^2BS-Ps require much less memory access than other algorithms in most cases. Taking the ACL series rule sets, for example, D^2BS only needs about 10-30 percent memory access of other algorithms, which indicates that D^2BS requires less time on accessing the memory.

We notice that D^2BS-Ps needs more memory accesses on FW5K and FW10K rule sets than other rule sets. This result may because the *P-Function* used in D^2BS-Ps tries to optimize the spatial performance, with the cost of more memory accesses. We also notice that HiCuts needs the most memory accesses in the average case. This is because the linear searching inside HiCuts trie's leaf nodes increases the access times a lot. Although D^2BS also adopts linear searching inside its *S-Blocks*, the sizes of the *S-Blocks* are kept small because of the effective partition by the *M-Vector*. Taking ACL10K rule set for example, the average size of *S-Blocks* is only around 10 when the *M-Vector* size is set to 4,096 (12 bits *E-Bits* are utilized).

4.1.2 Memory Access of Worst Case

The memory access of the worst case is usually used to evaluate the robustness of one algorithm's performance. Many algorithms may work well on average, but the performance will descend sharply on the worst case. The comparison results are given in Fig. 3b. As the figure shown, D^2BS-Pt still keeps the least memory accesses in most rule sets compared with other algorithms, while D^2BS-Ps also performs well in all rule sets except for FW5K and FW10K. That's also because that for D^2BS-Ps, we let the algorithm try to optimize for storage first, which may introduce some cost on memory accesses.

Comparing memory access times of the average case and the worst case, some interesting results are observed. For example, heuristic partition-based algorithms such as HiCuts keep similar largest access times on almost all rule sets, which means its time performance is not good but steady. While for D^2BS-Ps, it presents small access times except for very large rule sets including FW5K and FW10K. This is because to achieve good storage performance, D^2BS-Ps smartly optimizes its data structures to be more space-effective with the tradeoff on temporal performance. On both cases, D^2BS-Pt performs the best results on most rule sets, which indicates the effectiveness of our dynamic design.

4.1.3 Memory Storage Usage

Memory storage is also quite important, as most of today's platforms only have small fast accessing memory. Some platforms such as FPGA even have more strict memory limitations. The comparison results are shown in Fig. 4. In the figure, with logarithmic scale Y-axis, we can see that the memory used by D^2BS-Ps is at least an order of magnitude less than that of other algorithms for most of the rule sets, especially for large ones, for example, ACL10K, FW10K, and IPC10K. Note that on rule sets FW5K and FW10K, HiCuts cannot calculate out the final data structures on our platform; thus, we give the theoretical number of storage required for comparison. Also take rule set FW10K, for example, HSM, HiCuts, and HyperSplit require more than 50-MB memory while D^2BS-Ps only requires less than 1 MB. Even D^2BS-Pt, whose *P-Function* aims to optimize the temporal performance, also performs well in the spatial

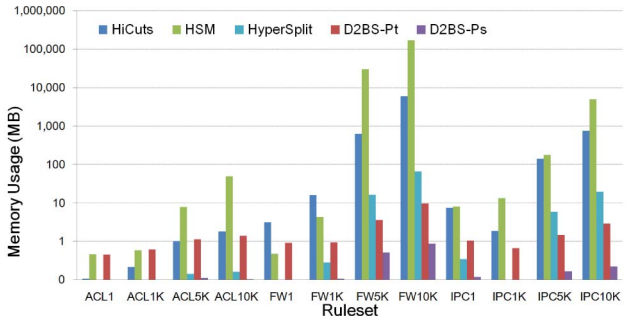


Fig. 4. Comparison of the memory requirement between D^2BS and other well-known algorithms.

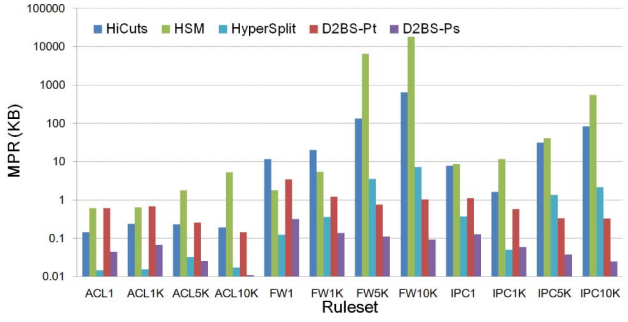


Fig. 5. Comparison of the scalability on spatial performance between D^2BS and other well-known algorithms.

performance. Although with some rule sets such as ACL series rule sets, HiCuts and HyperSplit can obtain a little better space performance than D^2BS-Pt, they both require much more storage than D^2BS-Pt in other large rule sets.

We also notice that the memory required on FW rule sets by all the algorithms is larger than that on ACL and IPC ones. This implies that FW rule sets have more complicated inherent structures. Thus, with the same rule set size, FW ones will generate more and complicated subspaces in the classification space than ACL and IPC ones. Therefore, we suggest that FW rule sets be suitable candidate to evaluate the real performance of the packet classification algorithm. Specially, with the largest FW rule set (FW10K), both D^2BS-Ps and D^2BS-Pt require far less memory than other algorithms.

4.1.4 Scalability

The scalability of the performance is another important evaluation factor as the size of rule set is growing larger quickly. We measure the scalability through the ratio of MPR. If this ratio jitters sharply among different types and sizes of rule sets, it implies that the algorithm's performance will be not stable with larger and more complicated rule sets in future; even it can perform well on small and simple rule sets.

In Fig. 5, we compare D^2BS's scalability with other three algorithms, which shows that both D^2BS-Pt and D^2BS-Ps perform better scalability than all the other algorithms. Among all 12 rule sets, MPR of HSM increases over 30,000 times from the best to the worst result, while the same results are over 4,500 for HiCuts and 480 for HyperSplit. However, for our algorithm (both D^2BS-Pt and D^2BS-Ps), MPR only increases about 20 times. This novel scalability will facilitates D^2BS perform well even for future's various rule sets.

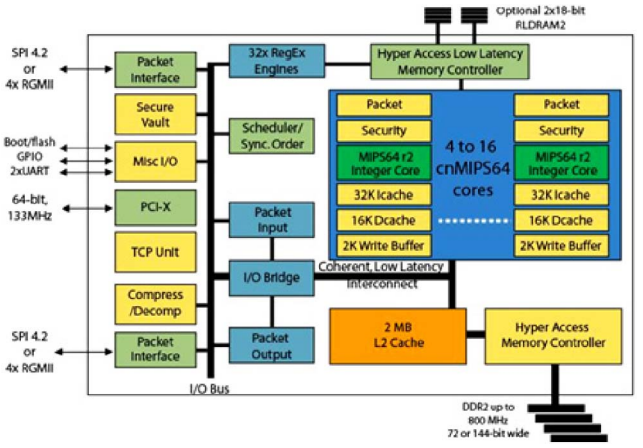


Fig. 6. Architecture of the Cavium OCTEON CN58XX platform.

Comparing HSM (best space partition-based algorithm) with HiCuts and HyperSplit (best rule set partition-based algorithm), we can see that the space partition-based algorithms show worse scalability than those based on the rule set partition based ones. This is because space partition algorithms build their data structure without considering the inherent characteristics of rule set. Even for rule sets with the same size but different inherent characteristics, performance of the space partition algorithms may still jitter a lot. In summary, space partition-based algorithms are more sensitive on the inherent complexity of the rule sets. Based on this observation, we can evaluate the complexity of rule sets with the same size but different types. Taking all the 10K size rule sets, (ACL, FW, IPC) for example, Fig. 5 illustrates that the highest MPR is obtained on the FW10K rule set, which means FW rule sets own more complicated complexity, and are more difficult to be classified.

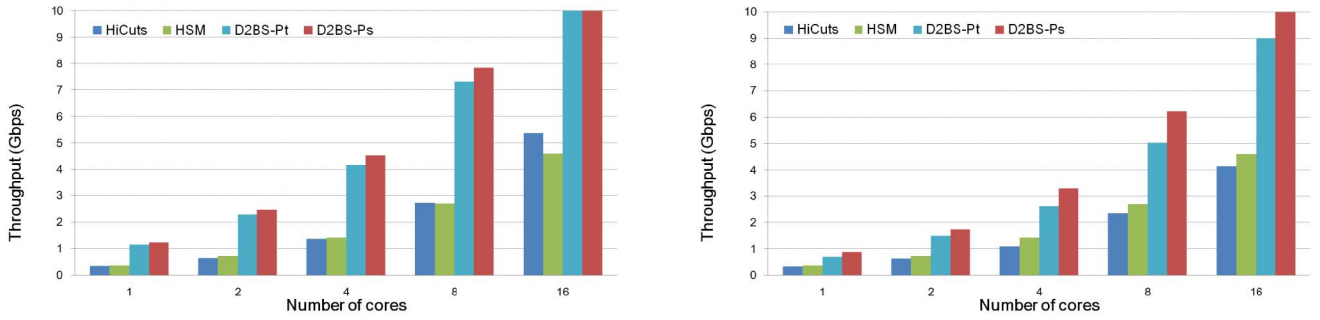
On the other hand, comparing the trend of MPR between D^2BS and other algorithms on rule sets from small to large ones, only D^2BS performs a clear descending on MPR. This result implies that only D^2BS achieves a sublinear storage requirements with the rule set size. This elegant property also promises that D^2BS is quite practical to process complicated rule sets.

4.2 Multicore NP Implementation

To evaluate the throughput performance of our algorithm on popular NP platform, we also implement and test D^2BS with one of the largest rule set, ACL10K, on the Cavium OCTEON CN5860 multicore platform. Our program runs in Simple Executive (SE) mode, thus we can evaluate the exact throughput performance of the algorithms. The block diagram of Cavium OCTEON CN5860 is shown in Fig. 6, and up to 16 cnMIPS64 cores can be utilized in parallel.

Figs. 7a and 7b show the throughput results of the average case and the worst case, respectively. From these results, as we expected, both D^2BS-Ps and D^2BS-Pt gain the best throughput performance over other algorithms from 1 core to 16 cores. Typically, D^2BS's performance are about 200-300 percent of that of other algorithms. Specially, with 64-byte packets[2] and 16 cores, only D^2BS

2. 64 byte is usually used as the minimal packet size for evaluation.



(a) Comparison of the average-case throughput.

(b) Comparison of the worst-case throughput.

Fig. 7. Comparison of the throughput of the average and the worst case on the NP platform.

achieves full 10-Gbps throughput in the average case. While in the worst case, D^2BS-Ps is also the only one that achieves full 10-Gbps throughput with 16 cores. All these results prove D^2BS a practical algorithm with the best performance for NP platform. Comparing D^2BS-Ps with D^2BS-Pt, we notice that the performance of D^2BS-Ps is a little higher than that of D^2BS-Pt. This may suggest that on NP platforms, algorithms should be designed to achieve efficient spatial performance.

4.3 FPGA-Based Implementation

To evaluate D^2BS on FPGA, the algorithm has been implemented on the Xilinx Virtex-5 FPGA. The FPGA (model: XC5VSX240T) has 2,048 Kb of Block RAM (BRAM). Considering the hardware limitation, we take D^2BS-Ps for the evaluation. The search algorithm has been pipelined to achieve high throughput. By pipelining, this design can classify one packet per clock cycle. The general architecture of the design is shown in Fig. 8.

The pipeline consists of two phases. The first phase involves masking out the bits from the packet header and using the masked bits as a pointer to the *S-Block* stored in the BRAM. To take advantage of parallelism of the BRAM within the FPGA, *S-Blocks* have been stored horizontally so that all the entries of an *S-Block* is read and matched at the same time. One entry of the *S-Block* is stored in one BRAM. Making the number of parallel BRAM equal to the size of the largest *S-Block*, we can then guarantee being able to match an *S-Block* within one clock cycle; hence, this step is always implemented in one pipeline stage. The second phase involves finding and selecting the highest priority rule from all the possible matched rules within the *S-Block*. Depending on the size of the *S-Blocks*, this step may involve multiple pipeline stages.

Within the actual hardware implementation, 10 match blocks are placed into a group and 10 groups are arranged into a cluster. Each group has its own rule selector that selects the highest priority rule from the match blocks and each cluster has its own rule selector to select the highest priority the groups. So for a 100 match block design, there will be 10 groups and 1 cluster. The second phase is then pipelined into two pipeline stages. The overall pipeline length is 3 for this design.

The performance of the FPGA implementation is shown in Fig. 9 and Fig. 10. Even with a 10K ACL rule set and 64-bytes packets, the FPGA design is capable of running at 131.987 MHz which is equivalent to 135 Gbps. This represents a 14 percent increase in bandwidth compared to the best FPGA design today [15]. For a packet classifier, the latency of the classification is just as important as the throughput. A short pipeline is desired because the shorter the pipeline, the lower the latency of the packet classifier. Compared to other FPGA-based packet classifier [14], [15], D^2BS's pipeline is significantly shorter. The latency is also reduced by four times, from 12 to 3 clock cycles.

All these results promise that D^2BS is naturally well suited to the FPGA architecture. Although masking bits from the packet header is difficult on normal processor, this is very simple on FPGA. Because each BRAM within the FPGA is separately addressed, the FPGA implementation does not require an index to find the *S-Block*. This, furthermore, reduces memory usages as well as a task required to translate the masked bits into an address. Finally, the parallelism offered by the FPGA allows the *S-Block* to be read and matched in parallel in one clock cycle.

4.4 Probability Distribution of E-Bits

The selection of *E-Bits* is important to the performance of our algorithm. With few well-selected *E-Bits*, the rule set can be partitioned effectively; while with badly selected ones, the rule set may be partitioned poorly. Thus, the distribution of *E-Bits* can illustrate the inherent characteristics of rule sets. In Figs. 11a and 11b, we show the probability distribution and the respective cumulative distribution function (CDF) of the *E-Bits* locations on

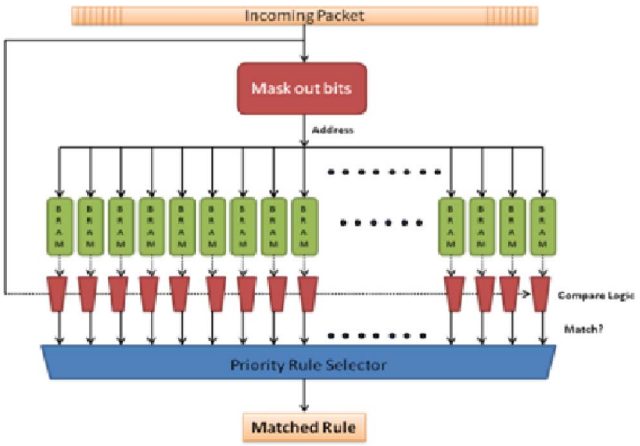


Fig. 8. Architecture of the system implementation on FPGA platform.

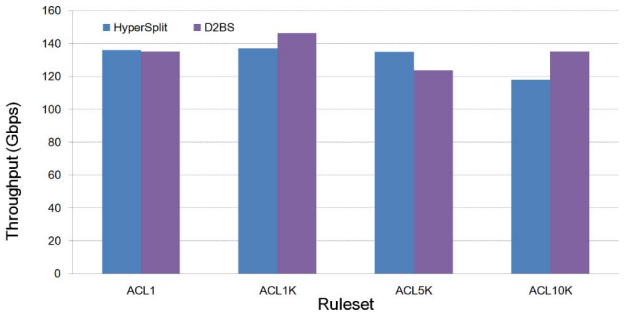


Fig. 9. Comparison of the throughput between D^2BS and HyperSplit on FPGA platform.

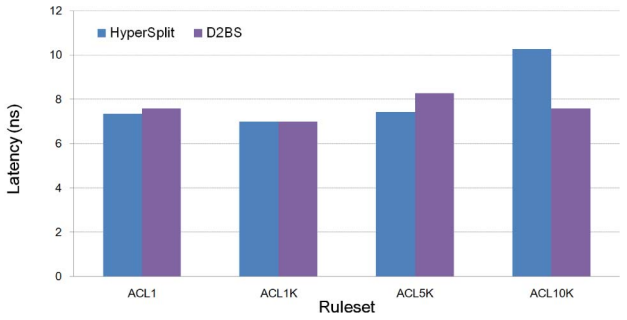


Fig. 10. Comparison of the latency between D^2BS and HyperSplit on FPGA platform.

different types of rule sets (ACL, FW, and IPC), among the total 104 bits (bit 0-103) in the packet header. In the evaluation, we choose 16 *E-Bits* on each rule set.

From the distribution on the ACL rule sets, we can see that most *E-Bits* occur at bit 0-3 (the start of source IP address field), bit 26-30 (the end of source IP address), bit 32-39 (the start of destination IP address) and bit 101 (within the protocol field), while both source port and destination port fields only provide few *E-Bits*. For IPC rule sets, similar distribution is observed. Most *E-Bits* stay at the IP address fields and the protocol field, quite few ones are found at the port fields. However, the distribution of FW rule sets is more average. Besides, the IP address fields and the protocol field, the destination port field also provides numbers of *E-Bits*.

Since the locations of *E-Bits* means the dimensions that are effective for partition, *E-Bits* actually indicate the significance bits for the classification rule sets according to our definition, or which bits can represent the real inherent structures of rule sets with high probability. These results indicate that the IP address fields and the protocol fields are important for all classification rule sets, while the FW rule sets also care the destination port field. This conclusion can be understood with the intentions of designing different types of rule sets. Both ACL and IPC rule sets mainly limit the access of users through IP or basic protocols, while FW rule sets also need to provide more advanced control of filtering the applications in a granular degree.

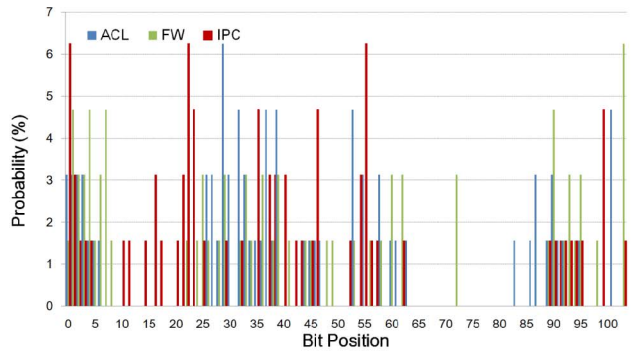
The global probability distribution and the global CDF on all rule sets are also given in Figs. 12a and 12b, respectively. From these figures, we can find the similar result. Most *E-Bits* occur at IP address fields, protocol fields, and destination port fields. One interesting observation is that for each field, a majority of *E-Bits* stay at the start and end parts, while few are located in the middle part. Taking the source IP address filed for example, the start 4 and end 4 bits provides 46.7 percent *E-Bits* of the whole 32 bits. Bits on the start part of field will cut the whole classification hyperspace into continuous subspaces, while bits on the end part will only influence smaller local subspace.

This characteristic implies that the global classification hyperspace can be partitioned effectively through two-level splitting: one is to split the global hyperspace into large continuous ones, and the other is to split at the bottom level subspaces. On the other hand, the results indicate the location of rules in the global classification hyperspace is quite uneven-distributed, which is also the foundation that intelligent algorithms designed based on rule sets' inherent characteristics such as D^2BS can perform better than other ones.

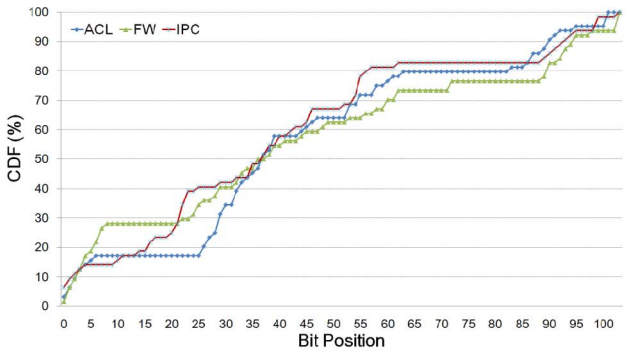
4.5 Discussion

All experimental results show that with granular heuristics, D^2BS works well in performance and scalability on various types and sizes of rule sets. At the same time, effective partitioning by dynamic data structures of D^2BS helps it perform flexibly with different requirements on diverse environments. This indicates that heuristic algorithms with dynamic data structures may get more advantages than traditional ones, especially for more complicated situations.

Comparing D^2BS-Pt and D^2BS-Ps, which are time-optimized and space-optimized, respectively, we can see

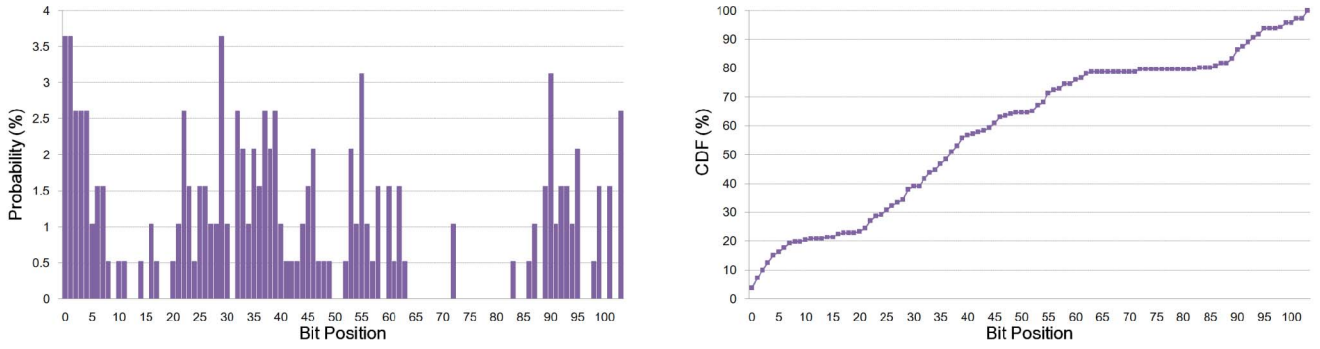


(a) Distribution probability of the *E-Bits* locations.



(b) CDF of the *E-Bits* locations.

Fig. 11. The probability distribution and the respective CDF of *E-Bits* location on different types of rule sets.



(a) Global distribution probability of the *E-Bits* locations.

(b) Global CDF of the *E-Bits* locations.

Fig. 12. The global probability distribution and the respective CDF of *E-Bits* locations among all rule sets.

D^2BS-Pt gains less memory access times but require more storage. Through results on NP platform, we can see D^2BS-Ps performs better than D^2BS-Pt in real throughput. This implies memory hierarchies especially cache in network processing hardware platforms, help to make algorithms with smaller storage requirement processing faster. Thus, for future memory hierarchical networking platforms, for example, multicore platforms, storage efficiency is really important for algorithms toward high processing performance.

Results on FPGA also prove the effectiveness of D^2BS. To the best of our knowledge, D^2BS is the first one that achieves over 135-Gbps throughput on Xilinx Vertex-5 FPGA platform with 10K size rule set. Leveraging pipelining and paralleling, our algorithm shows great capability in modern hardware platforms. This also promise D^2BS can even perform better in more powerful platforms.

5 Conclusion and Future Work

Technologies that try to achieve QoS via classifying packets have evolved rapidly over the last decade. However, practical solutions that can meet all requirements such as performance and scalability at the same time are still lacking.

In this paper, we propose a novel multituple packet classification algorithm called D^2BS. Unlike other existing solutions, D^2BS takes advantage of a dynamic heuristic partition of rule set at bit level, which allows it to better explore the inherent characteristics in rule sets and split rules more effectively. To meet different requirements under diverse situations, D^2BS utilizes a P-Function to optimize its data structure dynamically, which also promises a good flexibility.

To evaluate the performance of the proposed algorithm, we implement D^2BS on various types of platforms including IA-based platform, multicore NP, and FPGA, and compare it with existing well-known algorithms such as HiCuts, HSM, and HyperSplit. Experimental results show that D^2BS achieves superior temporal and spatial performance to existing algorithms, especially on very large rule sets. With 10K size ACL rule set and 64-byte packet, D^2BS achieves over 10-Gbps throughput on Cavium OCTEON CN5860 multicore NP, and over 135-Gbps throughput on Xilinx Vertex-5 FPGA, which is the best performance on the same platform. On the other hand, D^2BS also guarantees good scalability with the size of rule sets. Compared with other well-known algorithms, D^2BS is the only one that shows sublinear storage requirement with rule set size. All results promise D^2BS to be a practical algorithm with even larger and more complicated rule sets in future.

We also notice D^2BS's access time in worst case is several times larger than that in average case for some rule sets, which may decrease the storage utilization on FPGA platform. We are trying to design more effective storage structures on FPGA, which will even improve D^2BS's scalability for quite large rule sets in future. Our future work includes finding more effective heuristics and designing more efficient data structures, to improve the performance of D^2BS with other types of advanced platforms. Other interesting topics include the optimal results of the *E-Bits* selection, and the relationship between the *D-Table* construction and the partition.

Acknowledgments

The authors would like to thank all the reviewers for their insightful suggestions and comments.

References

[1] R. Kompella, K. Levchenko, A. Snoeren, and G. Varghese, "Every Microsecond Counts: Tracking Fine-Grain Latencies with a Lossy Difference Aggregator," *Proc. ACM SIGCOMM*, pp. 255-266, 2009.

[2] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "DPPC-RE: TCAM-Based Distributed Parallel Packet Classification with Range Encoding," *IEEE Trans. Computers*, vol. 55, no. 8, pp. 947-961, Aug. 2006.

[3] M. Overmars and A. van der Stappen, "Range Searching and Point Location among Fat Objects," *Proc. Second Ann. European Symp. Algorithms (ESA)*, pp. 240-253, 1994.

[4] D.E. Taylor and J.S. Turner, "ClassBench: A Packet Classification Benchmark," *IEEE/ACM Trans. Networking*, vol. 15, no. 3, pp. 499-511, June 2007.

[5] "OCTEON Plus CN58XX Multi-Core MIPS64 Based SoC Processors," Cavium Networks, http://www.caviumnetworks.com/OCTEON-Plus_CN58XX.html, 2013.

[6] "Virtex-5 FPGA Family," Xilinx, http://www.xilinx.com/products/virtex5/index.htm, 2013.

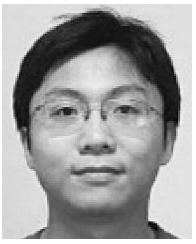
[7] B. Yang, X. Wang, Y. Xue, and J. Li, "DBS: A Bit-Level Heuristic Packet Classification Algorithm for High Speed Network," *Proc. 15th Int'l Conf. Parallel and Distributed Systems*, pp. 260-267, 2009.

[8] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.*, pp. 147-160, 1999.

[9] B. Xu, D. Jiang, and J. Li, "HSM: A Fast Packet Classification Algorithm," *Proc. 19th Int'l Conf. Advanced Information Networking and Applications,* 2005.
[10] P. Gupta and N. Mckeown, "Packet Classification Using Hierarchical Intelligent Cuttings," *Proc. Seventh Hot Interconnects Symp.,* pp. 34-41, 1999.
[11] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.,* pp. 213-224, 2003.
[12] T. Sherwood, G. Varghese, and B. Calder, "A Pipelined Memory Architecture for High Throughput Network Processors," *Proc. Int'l Symp. Computer Architecture,* pp. 288-299, 2003.
[13] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet Classification: From Theory to Practice," *Proc. IEEE INFOCOM,* pp. 648-656, Apr. 2009.
[14] W. Jiang and V. Prasanna, "Large-Scale Wire-Speed Packet Classification on FPGAs," *Proc. ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays,* pp. 219-228, 2009.
[15] Y. Qi, J. Fong, W. Jiang, J.L.B. Xu, and V. Prasanna, "Multi-Dimensional Packet Classification on FPGA: 100 Gbps and Beyond," *Proc. Int'l Conf. Field-Programmable Technology,* 2010.
[16] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching," *ACM SIGCOMM Computer Comm. Rev.,* vol. 28, no. 4, pp. 191-202, 1998.
[17] P. Pudil, J. Novovicová, and J. Kittler, "Floating Search Methods in Feature Selection," *Pattern Recognition Letters,* vol. 15, no. 11, pp. 1119-1125, 1994.

Baohua Yang received the BS degree from the Department of Automation, Tsinghua University, Beijing, China, in 2007, where he is currently working toward the master and PhD degrees. He is currently a research assistant in the Network Security Research Laboratory. His research interests include algorithmic, optimization, and performance issues in computer networking and architectures. He is a member of the IEEE.

Jeffrey Fong received the BASc degree from the University of British Columbia, Canada, in 2009. He is currently working toward the master's degree in the Network Securities Lab, Tsinghua University, Beijing, China. His research interests include network securities, high-speed network processing platforms, and FPGA designs. He is a member of the IEEE.



Weirong Jiang received the bachelor's and master's degrees from Tsinghua University, Beijing, China, in 2004 and 2006, respectively, and the PhD degree in computer engineering from the University of Southern California, Los Angeles, in 2010. He is currently working in Juniper Networks. His research interests include network processing, traffic analysis, and FPGA designs. He is a member of the IEEE.

Yibo Xue received the BS and MS degrees in computer science from Harbin Institute of Technology, Heilongjiang, China, in 1989 and 1992, respectively, and the PhD degree in computer science from the Institute of Computing Technology of Chinese Academy of Sciences, Beijing, China, in 1995. He is a senior member of the China Computer Federation. His research interests include computer network and information security, computer architecture, and parallel computing. He has published more than 50 papers, and is a coinventor of three patents. He is currently a vice director at the Center for Microprocessor and SOC Technology and a professor at the Research Institute of Information Technology, Tsinghua University. He is a member of the IEEE.

Jun Li received the BS and MS degrees in automation from Tsinghua University, Beijing, China, and the PhD degree in computer science from New Jersey Institute of Technology, now New Jersey Science and Technology University. He is currently a professor and a dean at the Research Institute of Information Technology, and an executive vice dean of the School of Information Science and Technology, Tsinghua University. He was a board director of XML Global and is currently a board member or advisor to several venture capital and startup companies, including GigaDevice and Agate Logic. He is also a deputy director of the Tsinghua National Lab for Information Science and Technology. He is a coauthor of more than 40 papers, and coinventor of four patents. He is also a managing director of an angle fund Versatile Venture Capital. His research interests include network security, pattern recognition, and image processing. He has been a member of the IEEE since 1996.