This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2009 proceedings.

Packet Classification Algorithms: From Theory to Practice

Yaxuan Qi, Lianghong Xu and Baohua Yang
Department of Automation,
Research Institute of Information Technology (RIIT),
Tsinghua University, Beijing, China.
yaxuan@tsinghua.edu.cn
{xlh05, ybh03}@mails.tsinghua.edu.cn

Yibo Xue and Jun Li
Research Institute of Information Technology (RIIT),
Tsinghua University, Beijing China.
Tsinghua National Lab for Information Science and Technology (TNList), Beijing China.
{yiboxue, junl}@tsinghua.edu.cn

***Abstract*—During the past decade, the packet classification problem has been widely studied to accelerate network applications such as access control, traffic engineering and intrusion detection. In our research, we found that although a great number of packet classification algorithms have been proposed in recent years, unfortunately most of them stagnate in mathematical analysis or software simulation stages and few of them have been implemented in commercial products as a generic solution. To fill the gap between theory and practice, in this paper, we propose a novel packet classification algorithm named HyperSplit. Compared to the well-known HiCuts and HSM algorithms, HyperSplit achieves superior performance in terms of classification speed, memory usage and preprocessing time. The practicability of the proposed algorithm is manifested by two facts in our test: HyperSplit is the only algorithm that can successfully handle all the rule sets; HyperSplit is also the only algorithm that reaches more than 6Gbps throughput on the Octeon3860 multi-core platform when tested with 64-byte Ethernet packets against 10K ACL rules.**

Keywords-algorithm; classification; multi-core; performance

I. INTRODUCTION

It is invariably the ultimate goal to improve the efficiency and security of network operation in today's Internet world. Access control, traffic engineering, intrusion detection, and many other network services require the discrimination of packets based on the multiple fields of packet headers, which is called packet classification.

Although packet classification has been widely studied for a long time, researchers are still motivated to seek novel and efficient packet classification solutions due to the following reasons:

- **Continual growth of network bandwidth**: The explosion in demand for network bandwidth owes much to the growth in data traffic. Leading service providers report bandwidths doubling on their backbones about every six to nine months [1]. As a consequence, novel packet classification solutions are required to handle the exponentially increasing traffics on both edge and core devices.
- **Ever-increasing complexity of network applications**: Traditional packet classification is mainly employed by firewalls to screen unwanted traffics. With more and more network applications implemented in today's network devices, packet classification is widely used for various kinds of applications, such as service-aware routing, intrusion prevention and traffic shaping. Thus, novel solutions must be more intelligent to effectively handle multifarious types of rule sets without significant loss of performance.
- **Technology innovations of network systems**: New technologies, such as multi-core network processors [2] [3], provide us with unprecedented computing power, as well as highly integrated resources. So novel packet classification solutions must be well suited to advanced hardware and software technologies to break the bottleneck, providing the availability of these innovative technologies to a vast majority of customers.

Looking back upon existing work, we found that although a great number of packet classification algorithms have been proposed in recent years, most of them stay in mathematical analysis and/or software simulation stage, and few of them have been implemented in commercial products as a generic solution. The gap between theory and practice in existing work can be summarized according to the different research motivations:

978-1-4244-3513-5/09/$25.00 ©2009 IEEE

- **Mathematics-based solutions**: Some algorithms focusing on extensive mathematical analysis have been proposed and some of them are reported to have excellent temporal/spatial complexity. However, algorithms of this kind can hardly be found to have any implementation in real-life network devices. This is mainly because pure mathematical solutions often add special conditions to simplify the problem and/or omit large constant factors in the Θ(•) notation which might conceal the explicit worst-case bound [4].
- **Observation-based solutions**: These algorithms employ the statistical characteristics observed in rules to achieve more efficient solutions for real-life applications. These algorithms often work well with specific type of rule sets. However, because packet classification rules for different applications have diverse features [5], few algorithms are "smart" enough to fully exploit the redundancy lying in different types of rule sets to obtain stable performance under various conditions.
- **Hardware-based solutions**: Another kind of algorithms is proposed to accelerate packet classification based on application specific hardware. Although this kind of solutions such as FPGA/ASIC based algorithms often have extremely high performance, they suffer from poor scalability and portability. Moreover, hardware solutions often mean high R&D cost and long time-to-market.

Thus packet classification is still an important problem and there is a great need for novel solutions. The gap between theory and practice motivates our research. In this paper, we propose a novel packet classification algorithm and evaluate it on a new-generation multi-core network processor. Main contributions of this paper include:

- **Problem Analysis**: Packet classification problem is inherently hard due to high worst-case complexity. However in real-life applications, the complexity of packet classification is far from the worst-case scenario. Based on the comparison of two representative algorithms with three types of real-life rule sets, we summarize types of solutions of existing work.
- **A Novel Algorithm**: In this paper, we proposed a novel packet classification algorithm by combining the advantages of existing algorithms: rule-based space decomposition and local-optimized recursion. The proposed HyperSplit algorithm guarantees explicit worst-case classification speed and explores the data-redundancy in rule sets to reduce memory usage. The data structure of HyperSplit is also carefully designed for efficient storage and fast access.
- **Performance evaluation**: The proposed HyperSplit algorithm, as well as HiCuts and HSM, is implemented and evaluated on the Cavium Octeon3860 multi-MIPS-core platform. Experimental results show that: on a variety of real-life rule sets, HyperSplit achieves better overall performance than HiCuts and HSM in terms of classification speed, memory usage and preprocessing time. All the source code and rule sets will be publicly available to encourage more extensive research and more objective evaluation in this area.

The following sections of this paper are organized as follows: Section II introduces the packet classification problem and analyzes its theoretical and practical complexity; Section III compares two representative packet classification algorithms to summarize the generic solutions of existing work; the proposed algorithm HyperSplit is described in Section IV and evaluated in Section V; In Section VI, we state our conclusion.

II. PROBLEM STATEMENT

A. The Packet Classification Problem

The object of packet classification is to classify packets by applying a set of rules to the header fields of a packet. Each rule $\boldsymbol{R}$ has $\boldsymbol{F}$ components, and the $\boldsymbol{f^{th}}$ component of rule $\boldsymbol{R}$, referred to as $\boldsymbol{R[f]}$, is a range match expression on the $\boldsymbol{f^{th}}$ field of the packet header. A packet $\boldsymbol{P}$ is said to match a particular rule $\boldsymbol{R}$, if $\forall \boldsymbol{f} \in [1, \boldsymbol{F}]$, the $\boldsymbol{f^{th}}$ field of the header of $\boldsymbol{P}$ satisfies the range expression $\boldsymbol{R[f]}$. If a packet $\boldsymbol{P}$ matches multiple rules, the matching rule with the highest priority is returned.

B. Complexity in Theory

Mathematically, packet classification can be viewed as a point location problem in computational geometry. All possible values in the $\boldsymbol{F}$ fields of a packet header form an $\boldsymbol{F}$-*dimensional search space* $\boldsymbol{S}$, and each packet $\boldsymbol{P}$ can be viewed as a *point* located in $\boldsymbol{S}$. The expression $\boldsymbol{R[f]}$ of rule $\boldsymbol{R}$ refers to a range in the $\boldsymbol{f^{th}}$ dimension of the search space and all the ranges specified by $\boldsymbol{R}$ compose an $\boldsymbol{F}$ -dimensional *hyper-rectangle*. If a packet $\boldsymbol{P}$ matches a particular rule $\boldsymbol{R}$, the point represented by $\boldsymbol{P}$ will fall into the hyper-rectangle specified by $\boldsymbol{R}$.

According to [6] [7], the best bounds for point location in $\boldsymbol{N}$ non-overlapping hyper-rectangles are $\boldsymbol{\Theta(\log N)}$ time with $\boldsymbol{\Theta(N^F)}$ space; or $\boldsymbol{\Theta(\log^{F-1} N)}$ time with $\boldsymbol{\Theta(N)}$ space. Packet classification is made yet more complex due to rule overlapping: In the worst case, $\boldsymbol{N}$ overlapping rules yield up to $\boldsymbol{(2N+1)^F}$ non-overlapping hyper-rectangles. Therefore, the packet classification problem has extremely high theoretical complexity in the worst case (e.g. 1K 4-field rules can consume $\boldsymbol{N^F = 1024^4 = 2^{40} = 1000G}$ memory).

C. Complexity in Practice

Although the theoretical complexity tells us it is infeasible to design a single algorithm that can perform well in all cases, fortunately, packet classification problems in real-life applications have some inherent characteristics that can be exploited to reduce the complexity [7] [8] [9]. Based on the following two facts, packet classification can be optimized for real-life applications: On the one hand, the rule set is given (determined) before a packet classification algorithm is applied to generate the classifier (the data structure used for packet classification, such as a decision-tree). Therefore, the data redundancy of the rules is known to the algorithm and hence can be exploited to reduce the complexity of the problem. On the other hand, although different types of rules have various statistical characteristics [10], the complexity of real-life rules

TABLE I. THEORETICAL VS. PRACTICAL COMPLEXITY FOR REAL-LIFE RULES

Rule Sets	*# rules*	*# non-over ranges in each field (theoretical)*	*# non-over ranges in sIP (practical)*	*# non-over ranges in dIP (practical)*	*# non-over ranges in sPT (practical)*	*# non-over ranges in dPT (practical)*	*# non-over rectangles (theoretical)*	*# non-over rectangles (practical)*
FW1	269	539	100	111	23	77	8.44×10^{10}	1.97×10^{7}
FW1-100	92	185	19	45	20	48	1.17×10^{9}	8.21×10^{5}
FW1-1K	791	1583	221	314	23	75	6.28×10^{12}	1.20×10^{8}
FW1-5K	4653	9307	3429	5251	23	77	7.50×10^{15}	3.19×10^{10}
FW1-10K	9311	18623	7270	13901	22	77	1.20×10^{17}	1.71×10^{11}
ACL1	752	1505	148	361	1	181	5.13×10^{12}	9.67×10^{6}
ACL1-100	98	197	48	120	1	70	1.51×10^{9}	4.03×10^{5}
ACL1-1K	916	1833	143	559	1	165	1.13×10^{13}	1.32×10^{7}
ACL1-5K	4415	8831	1157	1155	1	181	6.08×10^{15}	2.42×10^{8}
ACL1-10K	9603	19207	7921	1325	1	181	1.36×10^{17}	1.90×10^{9}
IPC1	1550	3101	271	226	59	94	9.25×10^{13}	3.40×10^{8}
IPC1-100	99	199	118	119	26	26	1.57×10^{9}	9.49×10^{6}
IPC1-1K	938	1877	559	796	49	78	1.24×10^{13}	1.70×10^{9}
IPC1-5K	4460	8921	886	2125	59	93	6.33×10^{15}	1.03×10^{10}
IPC1-10K	9037	18075	2377	4604	59	94	1.07×10^{17}	6.07×10^{10}

Note: sIP, dIP, sPT and dPT are source IP, destination IP, source Port and destination Port; FW, ACL, IPC are firewall policies, access control lists, and IP chain rules

is always far less than the theoretical worst case. Table I shows the theoretical and practical bounds of the number of non-overlapping hyper-rectangles for three types of rule sets with different number of rules. From the table we can see that:

- Different types of rules have various numbers of ranges on different packet header fields.
- The number of ranges on some fields is always far less than the worst-case bounds.
- The number of non-overlapping hyper-rectangles in practice is far less than the worst case.

Therefore, in theoretical analysis, the number of rules $\boldsymbol{N}$ and the number of fields $\boldsymbol{F}$ determine the worst-case complexity. While in practice, because the redundancy in the pre-defined rules is resolvable, more statistical and structural characteristics should be leveraged as heuristics to tune up performance. In conclusion, no perfect algorithm exists for all cases, but practical solutions can be found to solve real-life problems.

III. EXISTING WORK

A. Generic Solution

Due to the extremely high complexity, it turns out to be infeasible to solve the packet classification problem in a single step. Most of existing algorithms employ a *divide-and-conquer* solution: The search space is decomposed into sub-spaces, each of which is associated with a sub-set of rules. By recursively applying such space decomposition, the original problem is divided into a series of sub-problems with lower complexity. Generally, existing algorithms using the divide-and-conquer strategy resort to the following two steps:

- **Space Decomposition**: Given the search space $\boldsymbol{S}$, a field $\boldsymbol{f}$ and a set of rules $\boldsymbol{\mathcal{R}}$ as input, space decomposition divides $\boldsymbol{S}$ into multiple sub-spaces $\boldsymbol{S = S_1 \cup S_2 \cup ... \cup S_K}$ using a set of hyper-planes orthogonal to $\boldsymbol{f}$. Each sub-space $\boldsymbol{S_k}$, $(\boldsymbol{1 \le k \le K})$ is associated with a sub-set of rules $\boldsymbol{\mathcal{R}_k}$, where $\forall \boldsymbol{R} \in \boldsymbol{\mathcal{R}_k}, \boldsymbol{R \cap S_k \neq \emptyset}$.
- **Recursion Scheme**: Most of existing algorithms decompose the search space recursively in multiple stages. In the first stage, set $\boldsymbol{S^0}$ as the overall search space and $\boldsymbol{\mathcal{R}^0}$ as the whole rule set; after decomposition, $\{\boldsymbol{S_1^0, S_2^0, \dots, S_{K^0}^0}\}$ and $\{\boldsymbol{\mathcal{R}_1^0, \mathcal{R}_2^0, \dots, \mathcal{R}_{K^0}^0}\}$ are obtained. In the $\boldsymbol{i^{th}}$ stage, input each $\{\boldsymbol{S_k^{i-1}, \mathcal{R}_k^{i-1}}\}$ for all $\boldsymbol{1 \le k \le K^{i-1}}$ to further decompose the problem. The recursion is terminated as certain conditions are met, e.g. $\forall \boldsymbol{R} \in \boldsymbol{\mathcal{R}_k^{i-1}}, \boldsymbol{S_k^{i-1} \subseteq R}$ or $|\boldsymbol{\mathcal{R}_k^{i-1}}| \le \boldsymbol{T}$, where $\boldsymbol{T}$ is a predefined threshold.

Different implementations of these strategies are described and compared in the next sub-section.

B. Existing Algorithms

Among all the existing work, we focus on two algorithms: HSM (Hierarchical Space Mapping [11]) and HiCuts (Hierarchical Intelligent Cuttings [12]), because:

- HSM and HiCuts are representatives of two major categories of existing algorithms. Other existing work can be viewed as extensions or variations of these two algorithms [10] [13] [14] [15] [16] [17].
- A lot of experimental results of HSM and HiCuts are publicly available [5] [12] [13] [15], so that performance comparison with HiCuts and HSM will be more objective and convincing.

HSM performs hierarchically space mapping along all the $\boldsymbol{F}$ fields of the search space $\boldsymbol{S}$ in a specific order. HSM takes advantage of *rule-based* space decomposition strategy, i.e. $\boldsymbol{S}$ is decomposed along a predetermined field $\boldsymbol{f}$ by all the unique end-points of rule projections on $\boldsymbol{f}$. The recursion scheme of HSM is *impartially-treated*: sub-spaces obtained in the previous stage are further decomposed by all rules and along all other fields in a predetermined order. In the final stage, all sub-

Rule	*Priority*	*Field-X*	*Field-Y*
R1	1	[00,01]	[00,00]
R2	2	[00,01]	[00,11]
R3	3	[10,10]	[00,11]
R4	4	[11,11]	[11,11]
R5	5	[11,11]	[00,11]

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

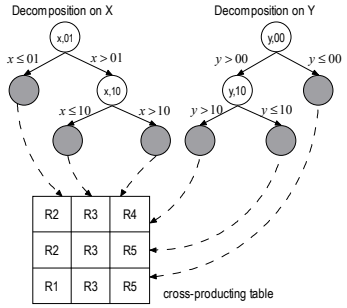

Figure 2. HSM for the example rules. Worst-case search requries 4 times of comparison.

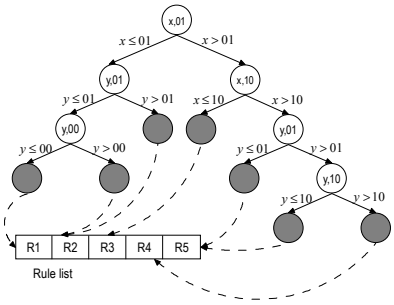

Figure 3. HiCuts for the example rules. Worst-case search requries 4 times of comparison.

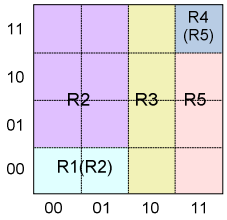

Figure 4. Geometric representation of the example rules shown in Figure 1.

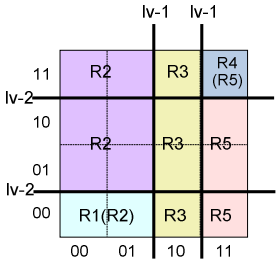

Figure 5. Space decomposition by HSM. The search space is decomposed into 9 sub-spaces in a 4-stage recursion.

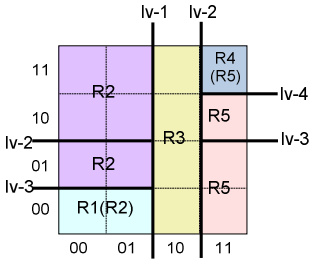

Figure 6. Space decomposition by HiCuts. The search space is decomposed into 7 sub-spaces in a 4-stage decomposition.

spaces are fully covered by the associated set of rules, in which the one with the highest priority is the best match.

In comparison, HiCuts employs the *equal-sized* space decomposition strategy: the search space $\boldsymbol{S}$ is decomposed (cut) into a certain number of equal-sized sub-spaces. The recursion scheme is also different from HSM. HiCuts adopts a *local-optimized* scheme for each stage of space decomposition: the field to apply decomposition depends on the rules associated with the current search space $\boldsymbol{S}$ instead of the whole rule set. HiCuts recursion terminates when the number of rules associated with current search space is less than a predetermined value (*binth* in [12]). The final results are obtained by linear search.

Figure 1 is an example rule set. There are 5 rules in a two dimensional search space. Some of the rules overlap with each other and the priority determines the final match in the overlapped regions. Figure 2 and Figure 3 show the fundamental ideas of HSM and HiCuts algorithms. HSM decomposes the search spaces based on rule projections along both the X and Y dimensions; HiCuts, however, uses a decision-tree to hierarchically decompose the search space with equal-sized cuttings. Figure 4~6 illustrate the different space decomposition in the 2-D search space.

C. Performance Analysis

The efficiency of packet classification algorithms can be evaluated according to three main metrics:

- Search speed: the overall time to classify an incoming packet, which is often measured by the number of memory accesses.
- Memory storage: the overall size of memory required to store the classification data-structure (e.g. a decision tree) generated by an algorithm.
- Preprocessing time: the overall time required by an algorithm to build the classifier.

Because $\boldsymbol{N}$ rules projected on each dimension generate at most $\boldsymbol{2N+1}$ end-points, and because HSM uses binary search on each of the $\boldsymbol{F}$ fields, the worst-case search time is then $\boldsymbol{F * log(2N+1)}$. While for HiCuts, because the number of cuttings and the field to decompose vary at different recursion stages, the worst-case search time of HiCuts is not explicit.

HSM and HiCuts use different methods to reduce the memory storage by exploring the data redundancy in real-life rules, although the worst-case storage is $\boldsymbol{\Theta(N^F)}$ for both. HSM employs a set of cross-producting tables to hierarchically map sub-spaces with identical rules to a single sub-space. HiCuts, however, aggregates consecutive sub-spaces using pointer arrays: sub-spaces with identical rules are associated to the same child-node for the next-stage decomposition.

When handling large rule sets, the space mapping operation in HSM incurs tremendous bit-map comparison between each pair of sub-spaces, resulting in long preprocessing time. The preprocessing of HiCuts is also time-consuming because of the exhaustive search among every possible number of cuttings

along all the dimensions to find the local-optimized decomposition scheme.

Experimental results of HSM and HiCuts are given in Section V.

IV. The Proposed Algorithm

A. Ideas

According to the analysis of existing work, novel packet classification algorithms should meet the following design goals:

- **Explicit search speed**: Because classification rate is often the most important performance metric in real-life systems, the search speed should be bounded in the worst case to guarantee the overall system performance.
- **Modest memory storage**: First and foremost, memory storage cannot exceed the overall system memory size; In addition, modest memory storage enables the use of fast memory technology, such as on-chip SRAMs to make memory access more efficient.

According to the above analysis, HSM well satisfies the first requirement in virtue of the deterministic temporal complexity, but it suffers from large memory usage. In comparison, although HiCuts has no explicit worst-case bound for search, it is storage-efficient due to the local-optimized recursion scheme. To foster the strengths and circumvent the weaknesses of existing work, the idea directly perceived is to combine the following two strategies:

- Rule-based space decomposition on each field to achieve deterministic worst-case search bound.
- Local-optimized recursion scheme to avoid unnecessary memory storage.

However, combination of these two ideas brings about a series of challenging problems in designing new algorithms:

- For the search speed: How to determine the explicit worst-case search bound no matter what kinds of rules are handled?
- For the memory storage: To decompose the field into $\boldsymbol{K}$ sub-spaces, rule-based partition needs to store $\boldsymbol{K-1}$ end-points. Then how to control the value of $\boldsymbol{K}$ to avoid the increase of the overall memory storage and the per-stage memory access during the search?

The answers to these questions are the key idea in our work. Our solution is to *apply binary space decomposition along optimized fields at each recursion stage*, then：

- Worst-case search time is explicit: $\boldsymbol{N}$ rules projected on each dimension yield no more than $\boldsymbol{2N+1}$ end-points, so if we use binary search on each of the $\boldsymbol{F}$ fields, the worst-case search time is $\boldsymbol{F*log(2N+1)}$, which is the supremum.
- Memory storage is efficient: Binary search needs only one end-point to compare in each stage, and for 5-tuple packet classification, one end-point is at most 32 bits. Thus the data-structure for search is compact and hence can be effectively stored and accessed.

Moreover, with binary decomposition, the preprocessing time is reduced: Because only two sub-spaces are generated after each partition, no space aggregation is needed. As a result, the most time-consuming part in the preprocessing of HSM and HiCuts is avoided.

B. HyperSplit

The proposed algorithm, HyperSplit, builds the classification data-structure using the following strategies:

1) Space Decomposition

Given the search space $\boldsymbol{S}$, a field $\boldsymbol{f}$ and a set of rules $\boldsymbol{\mathcal{R}}$ as input, HyperSplit applies the following rule-based decomposition strategy: First, project all rules in $\boldsymbol{\mathcal{R}}$ onto the $\boldsymbol{f}$ field, obtaining $\boldsymbol{M}$ $(\boldsymbol{2 \le M \le 2*N+1})$ end-points. Then sort these end-points in incremental order and store them in a temporary array $\boldsymbol{Pt[i]}$ $(\boldsymbol{1 \le i \le M})$. Every two consecutive end-points make up of a range segment, and there are totally $\boldsymbol{M-1}$ segments, denoted as $\boldsymbol{Sg[j]}$ $(\boldsymbol{1 \le j \le M-1})$. A hyper-plane orthogonal to the $\boldsymbol{f}$ field through one of the selected $\boldsymbol{M}$ end-points is used to split the current search space into two sub-spaces (this is why we use HyperSplit as the name of the proposed algorithm). A number of different heuristics can be adopted to select the end-point, and the following are three of them:

- Heuristic-1: Select $\boldsymbol{Pt\left[\left\lfloor M/2 \right\rfloor\right]}$. This heuristic indicates a *segment-balanced* decomposition, i.e. each sub-spaces contains the same number of segments along the $\boldsymbol{f}$ field.
- Heuristic-2: Select $\boldsymbol{Pt[m]}$, $\boldsymbol{s.t.}$ the number of rules overlapped with range $\boldsymbol{[Pt[1], Pt[m]]}$ is $\boldsymbol{\left\lfloor |\mathcal{R}|/2 \right\rfloor}$, i.e. half of the number of rules in $\boldsymbol{\mathcal{R}}$. This heuristic is *rule-balanced* because after decomposition, the number of rules in each sub-space tends to be the same.
- Heuristic-3: Assume there are $\boldsymbol{Sr[j]}$ rules in $\boldsymbol{\mathcal{R}}$ overlap with segment $\boldsymbol{Sg[j]}$. Select $\boldsymbol{Pt[m]}$, where $\boldsymbol{m}$ is the minimum value that satisfies $\boldsymbol{\sum_{j=1}^{m} Sr[j] > \frac{1}{2}\sum_{j=1}^{M} Sr[j]}$. This heuristic is named *weighted segment-balanced* strategy because $\boldsymbol{Sr[j]}$ can be viewed as the weight of segment $\boldsymbol{Sg[j]}$. Compared to Heuristic-1, the weighted scheme adds the overlapping information of $\boldsymbol{\mathcal{R}}$ in space decomposition, so the spatial complexity of each sub-space is balanced in a more effective way.

In our test, we choose the third heuristic due to its superior performance (saving about 50% memory storages).

2) Recursion Scheme

The selection of the local-optimized field to apply decomposition at each stage is the key issue for HyperSplit recursion. This goal can be achieved in multiple ways according to the different heuristics used in space decomposition:

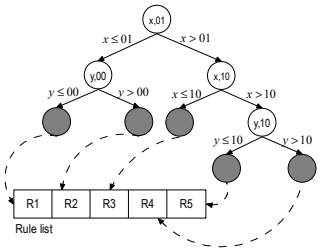

Figure 7. HyperSplit for the example rules. Worst-case search requries 3 times of comparison.

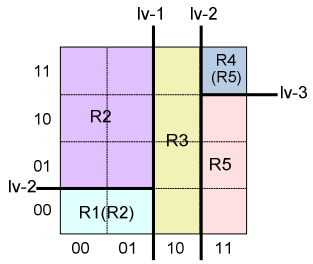

Figure 8. Space decomposition by HyperSplit. The search space is decomposed into 5 sub-spaces in a 3-stage recursion.

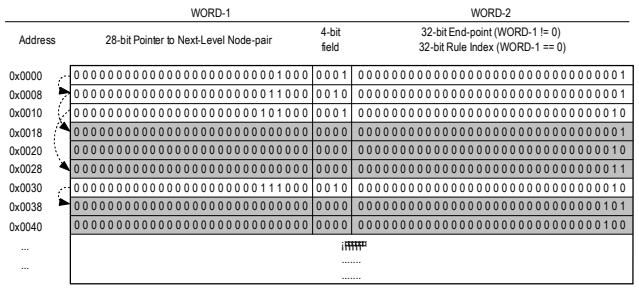

Figure 9. Compact data-structure of HyperSplit. (leaf-nodes are in grey)

- For Heuristic-1 and Heuristic-2, because fields with more end-points provide more candidates for local-optimized space decomposition, we can select the field with the largest $\boldsymbol{M}$ (number of end-points) to apply space decomposition at each stage.
- For Heuristic-3, each segment has a weight so we can select the field with minimum $\frac{1}{M}\sum_{j=1}^{M} Sr[j]$, i.e. the average weight of all the segments is minimized along the selected field.

At each recursion stage, HyperSplit associates each of the two sub-spaces with the sub-set of rules overlapping with it. The sub-space as well as the sub-set of rules is then set as input for the next-stage recursion. The recursion terminates when either of the following two conditions is met:

- The rule set $\mathcal{R}$ associated to the current search space $\boldsymbol{S}$ contains less then $\boldsymbol{T}$ rules, where $\boldsymbol{T}$ is a predetermined threshold. When $\boldsymbol{T} > 1$, linear search is used to find the final match.
- The current search space $\boldsymbol{S}$ is fully covered by all rules in $\mathcal{R}$. In this case, the rule with the highest priority in $\mathcal{R}$ is the final match.

Figure 7 illustrates the HyperSplit classifier for the example rule set shown in Figure 1. HyperSplit decomposes the search spaces using a decision-tree: at each internal node, the search space is decomposed into two sub-spaces along a local-optimized field; at each leaf node, the index of the matched rule is returned. Geometric representation of the space decomposition by HyperSplit is shown in Figure 8. Compared to HSM (Figure 5) and HiCuts (Figure 6), HyperSplit

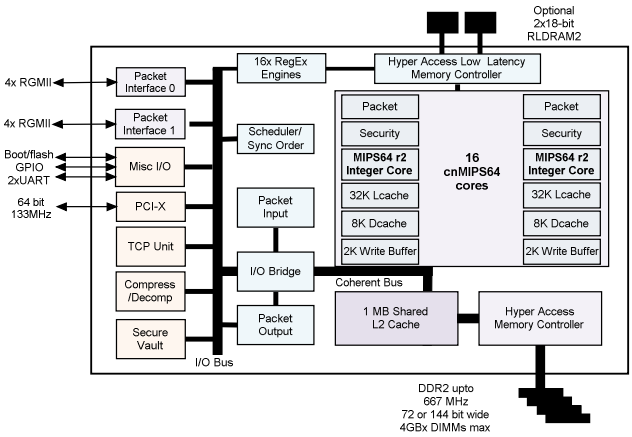

Figure 10. The Cavium OCTEON3860 multi-core platform [3].

decomposes the search space into the least number of sub-spaces, and at the same time, HyperSplit classify a packet with the minimum number of comparisons.

The classification data-structure of HyperSplit is shown in Figure 9. Both the internal node and leaf-node can be packed into a 64-bit long word for effective memory access. For each internal node, the first 32-bit WORD-1 contains a 4-bit field-to-decompose value (16 different fields supported) and a 28-bit address offset of the consecutively stored two child nodes (2^{28} 64-bit long words supported, i.e. 2GB memory space). The next 32-bit WORD-2 stores the end-point for decomposition. Leaf-nodes differ from internal nodes by setting WORD-1 as zero. When $\boldsymbol{T} = \boldsymbol{1}$, WORD-2 stores the index of the matched rule. When $\boldsymbol{T} > 1$, WORD-2 stores the pointer to locate the sub-set of rules for linear search.

V. PERFORMANCE EVALUATION

A. Data-set and Test-bed

In this section, the performance of the proposed HyperSplit algorithm, as well as HSM and HiCuts, is evaluated and compared using different types of rule sets on a popular multi-core platform. Because HyperSplit and HiCuts can use linear search to achieve different tradeoffs between memory storage and classification speed, there are actually 5 implementations for the 3 algorithms: HiCuts-1 (HiCuts with 1 rule in each leaf-node, i.e. without linear search), HiCuts-8 (HiCuts with up to 8 rules in each leaf-node), HyperSplit-1 (HyperSplit without linear search), HyperSplit-8 (HyperSplit with up to 8 rules in each leaf-node) and HSM.

The rule sets used in our test are publicly available [5], containing three types of rules: Access Control List (ACL), Firewall rules (FW) and IP Chains (IPC). Each rule set is named according to its type and size, e.g. FW1-10K refers to the first firewall rule set containing about 10,000 rules. All the rules are 5-dimensional: 32-bit source/destination IP addresses (represented as prefixes), 16-bit source/destination port numbers (represented as ranges) and 8-bit transport layer protocol (represented as discrete values).

Three primary measures of the performance, memory access, memory storage and preprocessing time, are obtained

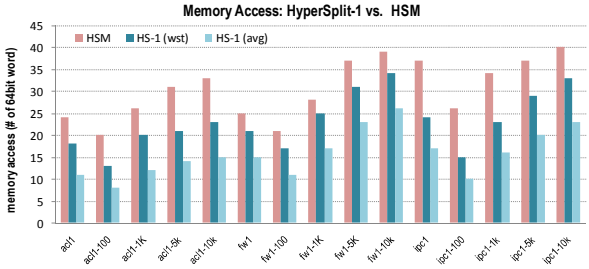

Figure 11. Memory Access: HyperSplit-1 vs. HSM

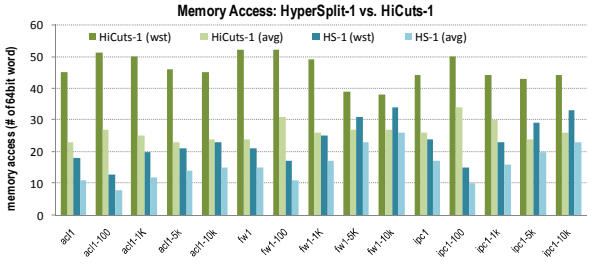

Figure 12. Memory Access: HyperSplit-1 vs. HiCuts-1 (without linear search at leaf-nodes)

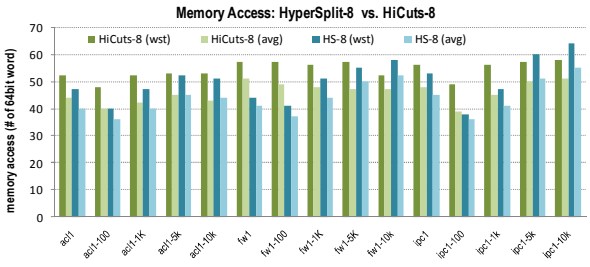

Figure 13. Memory Access: HyperSplit-8 vs. HiCuts-8 (with up to 8 times of linear search at leaf-nodes)

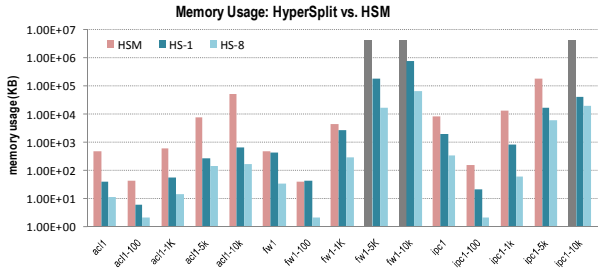

Figure 14. Memory Usage: HyperSplit vs. HSM (grey bars indicate running out of the 4GB memory)

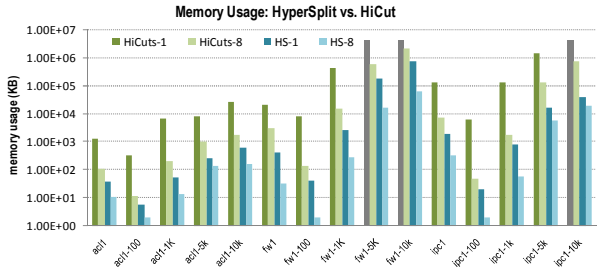

Figure 15. Memory Usage: HyperSplit vs. HiCuts. (grey bars indicate running out of the 4GB memory)

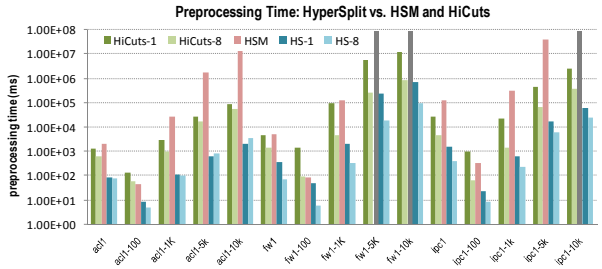

Figure 16. Preprocessing Time: HyperSplit vs. HiCuts and HSM (grey bars indicate running more than 24hours)

on a 2.0GHz dual-core PC with 4GB DDRII memory. All the code is written in C and compiled with -O2 optimization under Ubuntu 8.04LTS OS.

Throughput performance is tested on the Cavium Octeon3860 multi-core platform shown in Figure 10: processors are 16 MIPS cores running at 500MHz; network interfaces are eight 1Gbps RGMII ports; the memory hierarchy includes 1MB shared L2 cache and 2GB DDR2 SDRAM. Two programming modes are provided by Cavium Octeon Software Developer Kit (Cavium SDK version 1.5): programming in Linux mode (with Linux OS) or Simple Executive mode (no OS). Because packet classification is often considered as a fast-path application, all the algorithms are implemented in simple executive mode to achieve higher throughput.

B. *Test results*

1) *Memory access*

Worst-case and average-case memory accesses are illustrated in Figure 11~13. The unit of one memory access is a 64-bit long-word. Figure 11 compares the memory access of HyperSplit-1 and HSM, showing that the worst-case memory access of HyperSplit-1 is about 70% and the average access is less than 50% of that of HSM. We can see from Figure 12 that both the worst-case and the average-case memory accesses of HyperSplit-1 are about 50% less than HiCuts-1. While in Figure 13, the memory accesses of HyperSplit-8 and HiCuts-8 are nearly the same. This is because a large portion (about 70%) memory accesses are caused by the linear search, and hence the overall memory access seems not discriminating.

2) *Memory storage*

From Figure 14 and Figure 15 we can see that the memory storage of HyperSplit-1 is at least an order of magnitude less than that of HSM and HiCuts for most of the rule sets. HyperSplit-8 is even more memory-efficient, requiring only 1/10~1/100 of the memory of HyperSplit-1. For example, when tested on FW1-10K, both HSM and HiCuts-1 fail to build the classification data-structure due to exhaustion of memory usage (over 4GB). However, HyperSplit-1 successfully builds the binary tree with 753MB memory and HyperSplit-8 only consumes 66MB memory storages.

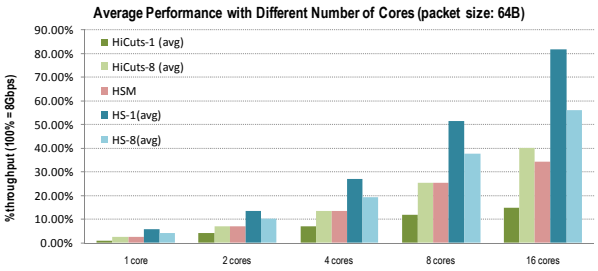

Figure 17. Average Throughput on Octeon3860. (rules: ACL1-10K, #cores: 16, packets: 64-1518 Bytes)

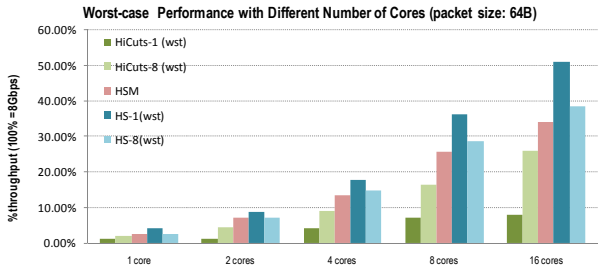

Figure 18. Worst-case Throughput on Octeon3860. (rules: ACL1-10K, #cores: 16, packets: 64-1518 Bytes)

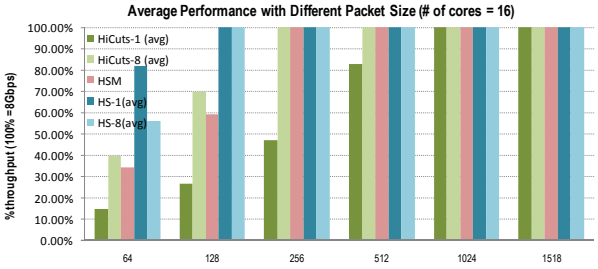

Figure 19. Speedups of Average Throughput on Octeon3860. (rules: ACL1-10K, #cores: 1-16, packets: 64 Bytes)

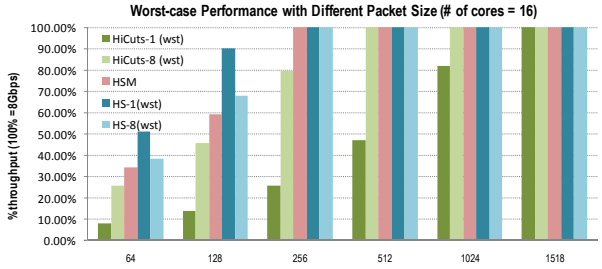

Figure 20. Speedups of Worst-case Throughput on Octeon3860. (rules: ACL1-10K, #cores: 1-16, packets: 64 Bytes)

From Figure 14 and Figure 15 we can also see that the memory usage of FW rule sets by all the three algorithms are much larger compared to ACL and IPC rules, indicating this type of rule sets have more complicated geometric structures. This is in accordance with the data provided by Table I in Section II: with the same number of rules, FW rules generate more non-overlapping hyper-rectangles than ACL and IPC rules in the multi-dimensional search space.

3) Preprocessing time

Preprocessing time of HyperSplit, HSM and HiCuts are shown in Figure 16. Compared to HSM and HiCuts, HyperSplit runs 10~100 times faster when handling the same set of rules. On small rule sets such as ACK1-100, HyperSplit requires less than 10 milliseconds in preprocessing. On large and complicated rule sets, e.g. FW1-10K, HyperSplit-8 requires 1.5 minutes and HyperSplits-1 requires 12.5 minutes for preprocessing. In comparison, HSM and HiCuts-1 spend more than 24 hours in preprocessing on FW1-10K.

4) Throughput on Octeon3860

Figure 17~20 shows the packet classification throughput and the relative speedups of HyperSplit, HSM and HiCuts on Cavium Octeon3860 multi-core platform. The input packets are 64~1518 Byte Ethernet packets and the tested rule set is ACL1-10K (the largest rule set that can be successfully processed by all the algorithms). Without loss of generality, header of each input packet generated by SmartBit is modified by core software in Octeon3860 to have a random value for classification. This randomization avoids the possible high cache hit-rate caused by the limited number of packet flows.

Figure 17 and Figure 18 show the average-case and worst-case throughput with different packet size. For the 64-Byte minimum Ethernet packets, HyperSplit-1 achieves 6.4Gbps throughput on average and 4Gbps in the worst case. In comparison, HSM and HiCuts reach less than 3.2Gbps in the average case and only 2.7Gbps in the worst case. The average throughput of both HyperSplit-1 and HyperSplit-8 reaches 100% (8Gbps) when input 128-Byte or lager packets. However, HSM and HiCuts-8 reach 100% throughput only when packet size is larger than 256. HiCuts-1 has the worst performance, reaching 100% only with 1024-Byte or larger packets.

Relative speedup is defined as the ratio between the throughput using multiple cores and that using a single core. In Figure 19 and Figure 20, as the number of cores increases, the throughput (for 64-Byte packets) of HyperSplit grows near linearly: for 16 cores, the speedup of HyperSplit-1 is 14.6 and the speedup of HyperSplit-8 is 13.6.

C. Discussions

Theoretically, the memory access of HyperSplit and HSM has the same worst-case bound. However, the test result in Figure 11 shows that the memory access of HyperSplit is significantly less than that of HSM. This is because HyperSplit decomposes the search space based only on the rules overlapping with the current search space, while HSM uses the overall rule set for space decomposition. So in practice, HSM always generates more end-points than HyperSplit along each field, resulting in more memory accesses for the binary search.

Compared to HiCuts, although HyperSplit partitions the search space only into two sub-spaces at each stage, the overall number of recursions (i.e. decision tree depth) is still smaller than that of HiCuts, and hence the memory access of HyperSplit is also less (Figure 12, 13). This indicates the advantage of the rule-based decomposition strategy: in practice, a single "smart" decomposition is often more effective than multiple "mean" cuttings.

In theoretical analysis, packet classification with more memory accesses is assumed to have lower classification rate due to the I/O latency. However, some of the test results are inconsistent with this assumption. For example, the throughput of HyperSplit-8 is higher than HSM (Figure 19, 20) although HyperSplit-8 requires more memory accesses (Figure 11). Similarly, HiCuts-8 obtains higher throughput than HiCuts-1despite its larger number of memory accesses. These experimental results expose the limitations of theoretical analysis. In practice, however, the actual number of system cycles required to process a packet is not proportional to the number of memory accesses, because memory access can take place at different types of memory, each of which has different size and latency. According to the memory hierarchy shown in Figure 10, because the memory usage of HyperSplit is less than 1MB, most of the memory accesses tend to take place in the L2 cache. In comparison, although the number of memory accesses of HSM is less than that of HyperSplit-8, the access to the large cross-producting tables (49MB for ACL1-10K) increases the overall latency of HSM. This explains why the throughput of HSM is lower than that of HyperSplit-8. Similarly, although HiCuts-8 requires more memory accesses than HiCuts-1, its memory storage is less than 1/10 of that of HiCuts-1 (1.9M vs. 27MB). So HiCuts-8 achieves a better classification rate due to the higher L2 cache hit-rate.

In our research, the basic ideas of HyperSplit originate from theoretical analysis while the design and implementation of HyperSplit take special care of practical issues. The binary decomposition scheme, the weighted segmentation heuristic as well as the compact node-structure, all contribute to the superior performance of HyperSplit in our test.

VI. CONCLUSION

In our research, we found that most existing packet classification algorithms stay in mathematical analysis or software simulation stages and few of them have been implemented in commercial products as a generic solution. To fill the gap between theory and practice, we design a novel packet classification algorithm by exploiting the advantages of existing work. Compared to the well-known HiCuts and HSM algorithms, the proposed HyperSplit algorithm achieves superior performance in terms of classification speed, memory usage and preprocessing time. The efficiency and practicability of the HyperSplit is verified by the test on Cavium Octeon3860 multi-MIPS-core platform. To encourage more extensive research and more objective evaluation, all the source code and rule sets will be publicly available on our website [21].

ACKNOWLEDGMENT

This work was supported by National High-Tech R&D 863 Program of China under grant No. 2007AA01Z468.

REFERENCES

[1] http://www.cisco.com/univercd/home/home.htm

[2] http://www.intel.com/design/network/products/npfamily/index.htm

[3] http://www.cavium.com/OCTEON_MIPS64.html

[4] Filippo Geraci, Marco Pellegrini, Paolo Pisati "Packet Classification via Improved Space Decomposition Techniques," in IEEE INFOCOM, 2005.

[5] http://www.arl.wustl.edu/~hs1/PClassEval.html

[6] M. H. Overmars and A. F. van der Stappen, "Range Searching and Point Location among Fat Objects," Journal of Algorithms, 21(3), 1996.

[7] P. Gupta and N. McKewon, "Algorithms for Packet Classification," IEEE Network, March/April, 2001.

[8] D. E. Taylor, "Survey & Taxonomy of Packet Classification Techniques," Technical Report, Washington University in Saint-Louis, USA, 2004.

[9] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar and A. T. Campbell, "Directions in Packet Classification for Network Processors," Proc. of the 2nd Workshop on Network Processors (NP2), 2003.

[10] S. Singh, F. Baboescu, G. Varghese and J. Wang, "Packet Classification Using Multidimensional Cutting," Proc. of ACM SIGCOMM, 2003.

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

[12] P. Gupta and N. McKeown, "Packet Classification Using Hierarchical Intelligent Cuttings," Proc. Hot Interconnects, 1999.

[13] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," Proc. ACM SIGCOMM, 1999.

[14] Y. X. Qi, B. Xu, F. He, B. H. Yang. J. M. Yu and J. Li, "Towards High-performance Flow-level Packet Processing on Multi-core Network Processors," Proc. of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2007.

[15] Y. X. Qi, B. Xu, F. He, X. Zhou, J. M. Yu, and Jun Li, "Towards Optimized Packet Classification Algorithms for Multi-Core Network Processors," Proc. of the 2007 International Conference on Parallel Processing (ICPP), 2007.

[16] D. Liu, B. Hua, X. Hu and X. Tang, "High-performance Packet Classification Algorithm for Many-core and Multithreaded Network Processor," Proc. of the 6th IEEE International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2006), 2006.

[17] T.Y.C Woo "A Modular Approach to Packet Classification: Algorithms and Results," in IEEE INFOCOM, 2000.

[18] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducting of field labels," in Proc. IEEE INFOCOM, Mar. 2005, pp. 269-280.

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

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

[21] http://security.riit.tsinghua.edu.cn/share/index.html