Scalable Packet Classification through Maximum Entropy Hashing

Lynn Choi[1], Jaesung Heo[1], Hyogon Kim[1], Jinoo Joung[2], Sunil Kim[3]

[1] The Department of Electronics and Computer Engineering, Korea University,
Anam-Dong, Sungbuk-Ku, Seoul, Korea
{lchoi, jsheo, hyogon}@korea.ac.kr
Tel: +82-2-3290-3249
Fax: +82-2-921-0544
[2] i-Networking Laboratory, Samsung Advanced Institute of Technology
Giheung-Eup, Yongin-Shi, Gyeonggi-Do, Korea
jjoung@samsung.com
[3] The School of Information and Computer Engineering, Hongik University
72-1 Sangsu-Dong, Mapo-Gu, Seoul, Korea
skim@cs.hongik.ac.kr

Abstract. In this paper we propose a new packet classification algorithm, which can substantially improve the performance of a classifier by decreasing the rulebase lookup latency. The algorithm hierarchically partitions the rulebase into smaller independent sub-rulebases by employing hashing. By using the same hash key used in the partitioning a classifier only needs to look up the relevant sub-rulebase to which an incoming packet belongs. For an optimal partitioning of rulebases, we apply the notion of maximum entropy to the hash key selection. We performed the detailed simulations of our proposed algorithm on synthetic rulebases of size 1K to 500K entries using real packet traces. The results show that the algorithm can significantly outperform existing classifiers by reducing the size of a rulebase by more than four orders of magnitude with just two-levels of partitioning. Both the space and time complexity of the algorithm exhibit linearity in terms of the size of a rulebase, suggesting a good scalable solution for the packet classification with a large rulebase.

1 Introduction

Packet classification is one of the most fundamental building blocks in many networking functions such as Diff-Serv traffic conditioning, firewall, VPN, traffic accounting and billing, load-balancing, and policy-based routing. These functions need to track *flows* and give the same treatment to the packets in a flow. A rulebase stores classification rules, which define the flows and their corresponding treatments. Since a flow is defined by the header values of a packet, a classifier's duty is to examine the header and identify the corresponding flow.

Internet traffic is not only fast growing, but it is also diversifying both in applications and in protocols. New applications and protocols such as Internet telephony, security protocols, and peer-to-peer applications are being rapidly deployed in addition to the traditional Internet applications such as Web, ftp, and email. As a result, the rulebase size is rapidly increasing. In a recent study [9], Woo argues that a rulebase with over a million entries is possible in future packet classification applications. From the classifier's viewpoint, this implies that for each packet the classifier must be able to find the matching rule with the highest priority amongst all the rules in the rulebase at the wire speed. Thus, there has been a renewal of interest [1, 2, 4, 5, 6, 7, 8, 10] in the scalability issue in terms of the size of a rulebase. Most of existing works, however, mainly focus on relatively small classifiers, e.g., with less than 20K rules [12]. To address this issue, we propose a new scalable packet classification algorithm that can scale well up to this size.

The motivation of our algorithm is based on the observation that a given packet matches only a few rules even in large classifiers [1]. This strongly implies that most of rules in any given rulebase are independent. Thus, we can partition the rulebase into many smaller independent sub-rulebases. As long as the matching sub-rulebase can be identified quickly, the performance of the rulebase lookup can be substantially improved since the lookup needs to be performed only in the final sub-rulebase. This is achieved by hierarchically decomposing the original rulebase into many smaller independent sub-rulebases based on the rules' definitions.

The algorithm is carried out in two phases: preprocessing and classification. First, during the preprocessing phase we hierarchically partition the original rulebase into many smaller independent sub-rulebases by hashing on the bit fields selected from the classification space. The degree of the partitioning depends on the density of a sub-rulebase in the classification space. The denser the sub-rulebase, the more partitioning is needed. This hierarchical partitioning stops until all the sub-rulebases are small enough. Then, during the classification phase a classifier inspects each incoming packet using the same hash key used in the preprocessing and identifies the sub-rulebase relevant to the packet. The search to find a matching rule is performed only in the final sub-rulebase where any existing lookup algorithm can be employed.

For an optimal partitioning of rulebases, we apply the notion of *entropy* in this paper, which guides us to choose the bits that most evenly divide the given rulebase. When the hash keys are selected to maximize the entropy, a rulebase is partitioned evenly into sub-rulebases under the smallest variance. As a consequence, we can achieve the smallest depth in the partitioning tree, which directly translates to the smallest number of hash table lookups. If the depth is small and the final sub-rulebase is small enough, we can achieve a low per-packet classification delay.

To evaluate the performance of our classification algorithm, we have applied our algorithm to real-life packet traces under synthetic rulebases of size 1K to 500K rules. The results show that the algorithm can reduce the size of the original rulebase by several orders of magnitude with only two-levels of partitioning, which requires only a couple of memory lookups. For example, a rulebase with 100K rules can be reduced to a sub-rulebase with only 7.6 rules on average and 258 rules in the worst case. In view of memory accesses, our algorithm requires 2 or 3 times less number of memory lookups compared to best classification algorithms known so far. Furthermore, the algorithm exhibits scalability in both its memory requirement and classification performance as we increase the size of a rulebase.

This paper is organized as follows. Section 2 defines the packet classification problem and presents the overall algorithm of our packet classification process. The section also introduces several partitioning algorithms including the notion of entropy and discusses the variations of the proposed algorithm to handle rule definitions with range and prefix mask description. Section 3 describes our experimentation methodology and summarizes the results. Section 4 concludes the paper.

2 The Proposed Classification Algorithm

2.1 Problem Definition

We can define the packet classification problem as follows. Given a *rulebase*, $R = \sum_{i=1}^{n} r_i$, which is a set of rules, a packet classifier needs to identify the rule that an incoming packet matches to by looking up one or more fields of the packet header. Each rule is specified by the range of values in one or more fields of a packet header. Specifically, in *d-dimensional packet classification*, each rule r_i is defined over d fields. Formally, r_i is defined by a tuple (C_i, A_i) where C_i is called a *classification space* and A_i is the associated action of rule r_i. The *classification space* is defined by the crossproduct, $C_i = F_1 \otimes F_2 \otimes \ldots F_d = \prod_{k=1}^{d} F_k^i$ where F_k^i is a range of values the field k must take. A rule r_i match a packet $p = \{b_1, b_2, .. b_d\}$ if for $\forall k, b_k \in F_k^i$ where b_k is a singleton. Multiple rules can match a packet. Thus, a classifier must identify the highest priority rule among all the matching rules. Intuitively, this requires the classifier to lookup the header fields of an incoming packet and to compare them against the rules in the rulebase one by one in order of decreasing priority. When n, i.e. the number of rules, is large or the arrival rate λ of incoming packets is high, this is a time-consuming serial process, which will limit the speed of the classifier. Thus, the essence of the problem is to find a fast yet scalable classification function both in time and in space.

2.2 Proposed Algorithm

Our classification algorithm is based on the conjecture that, in a rulebase, only a few rules have the possibility of matching a given packet. Let's look at the rulebase example of a typical firewall [15] shown in Table 1 where inner network serves several application services such as HTTP, telnet and FTP. Rules ***R1, R2,*** and ***R3*** represent grant of these connection requests while ***R0*** protects inner network against spoofing attacks. ***D*** is the default deny rule for all other communications. The protocol field in Table 1 suggests that a packet using UDP protocol can be matched only to ***R0*** or ***D***. Thus, ***R1***, ***R2*** and ***R3*** need not be matched against a UDP packet.

The algorithm consists of two phases: preprocessing and classification. The idea is to use divide-and-conquer approach. First, during the *preprocessing phase* we divide the original rulebase into many smaller independent sub-rulebases based on the values of classification fields where each rule is defined. Then, during the *classification phase* a classifier looks up the same header fields of an incoming packet and identifies the sub-rulebase where the relevant rules are stored. Thus, when the ratio (s/n) of the size of the sub-rulebase (s) over the size of the original rulebase (n) is small, then we can overcome the scalability issue by a single memory lookup to the hash table, which is constructed during the preprocessing phase. If a sub-rulebase is still large, then the sub-rulebase can be re-partitioned until the final sub-rulebase is small enough.

Table 1. A rulebase example of a firewall. †Inner side: protected local network by the firewall. ‡Outer side: network separated from inner side network by the firewall

Rules	Protocol	Src. Port	Dst. Port	Src. IP	Dst. IP	Action	Description
R0	*	*	*	Inner side†	Inner side†	**Deny**	Protection against Spoofing Attacks
R1	TCP	1024~65535	80	Outer side‡	Inner side†	**Accept**	HTTP Service
R2	TCP	1024~65535	23	Outer side‡	Inner side†	**Accept**	Telnet Service
R3	TCP	1024~65535	21	Outer side‡	Inner side†	**Accept**	FTP Service
D	*	*	*	*	*	**Deny**	Default Rule

According to [12], a rule $r_1 = (\prod_{k=1}^{d} F_k^1, A_1)$ *overlaps* with a rule $r_2 = (\prod_{k=1}^{d} F_k^2, A_2)$ if $\forall k\, F^1_k \cap F^2_k \neq \varnothing$. Intuitively, two rules overlap if there exists any instance of a packet that matches both rules. Since sub-rulebases differ at least in those bits that are selected as the hash key, a packet cannot match both sub-rulebases at the same time. Thus, the independence among sub-rulebases is guaranteed. Therefore, we need to look up only the relevant sub-rulebase after inspecting a packet on the same bit fields.

2.2.1 Preprocessing Phase: Rulebase Partitioning and Hash Table Construction

In preprocessing phase, we partition the original rulebase into many independent sub-rulebases. For example, the rules governing HTTP, FTP, and SMTP traffic can be partitioned into separate sub-rulebases. Then, by looking up the protocol field of an incoming packet, we only need to look up the sub-rulebase with the same protocol.

We can choose any of the bits in the classification fields as a hash key. If we select 8 bits, then we create a hash table with 2^8=256 entries, each of which points to a sub-rulebase. Intuitively, two rules may overlap if they map to the same sub-rulebase while rules mapped to different sub-rulebases would never overlap, which implies that they are independent. Sub-rulebases larger than a threshold value, such as 16 rules, can be repartitioned with another hash key, which must be different from the first hash key. This hierarchical partitioning stops until all the sub-rulebases are small enough. However, our experimentation results show that two levels of partitioning are enough for a rulebase under 500K rules.

Both the space and time complexity of our classification algorithm depend on the number of nodes and the depth of the partitioning hierarchy. To reduce the number of partitioning we need to partition a rulebase into sub-rulebases as evenly as possible so that the number of empty sub-rulebases is minimized and the number of rules in sub-rulebases must follow uniform distribution. This partitioning efficiency depends on the hash key selection algorithm, which we will discuss in detail in Section 2.3.

Table 2. A rulebase example and its hash tables. * denotes a don't care bit.

Classification space ($b_0b_1b_2b_3.........b_{103}$)	**Rule**
0000 0110……......	**R0**
0000 0110………..	**R1**
0001 0001………..	**R2**
0000 0001…….….	**R3**
**** ****………..	**D**

Hash Key 8MSBs ($b_0b_1b_2b_3b_4b_5b_6b_7$)	**Sub-rulebase**
0000 0000	*Null Entry*
0000 0001	**R3**
:	*Null Entry*
0000 0110	**R0, R1**
:	*Null Entry*
0001 0001	**R2**
:	*Null Entry*
1111 1111	*Null Entry*

Hash Key b_3b_5	**Sub-rulebase**
00	**R3**
01	**R0, R1**
10	**R2**
11	*Null Entry*

2.2.2 Classification Phase

After we partition a rulebase and construct hash tables during the preprocessing stage, a classifier can narrow down the rulebase lookup by mapping an incoming packet into the corresponding sub-rulebase where the packet can be applied. The classifier looks up the hash table by using the hash key extracted from the packet header.

Let us consider the rulebase example shown in Table 2. Assume that rules ***R0*** to ***R3*** are listed in the decreasing order of priority. We assume 5-dimensional classification, which uses 104-bit fields from protocol (8), source port (16), destination port (16), source (32) and destination (32) IP addresses from the header.

In Table 2, we show only 8 most significant bits (MSBs) of a classification space, which may represent any header field such as the protocol. We partition the rulebase to 256 buckets by using the 8 MSBs and create the hash table as shown in Table 2. Rules in one sub-rulebase do not overlap with rules in other sub-rulebases. When a packet arrives, the classifier extracts the 8 MSBs from the header and uses it as an index to the hash table. If the hash table entry is not empty, then the classification is performed within the sub-rulebase. Otherwise, the default rule is the matching rule.

2.3 Hash Key Selection

2.3.1 The First-Level Partitioning

For the first-level partitioning, we only consider the protocol and port numbers as a hash key since these fields can naturally classify rules based on the Internet services governed by the rules. For example, HTTP service corresponds to protocol 6 and server port 80. The classification space at this level is comprised of protocol (8), source (16), and destination port (16) numbers. To limit the size of the hash table, we select a subset from the classification space as a hash key. In our implementation, we use a 17-bit hash key, which suggests a hash table with 128K entries. There is a tradeoff between the memory space and the depth of the partitioning hierarchy depending on the size of the hash key. Assuming each entry contains either a 32b address or NULL-pointer, the size of the table is 512Kbytes.

For the 17-bit hash key, we first select 6 bits from the protocol field using the entropy-maximizing key selection algorithm, which we will discuss in detail in Section 2.3.2. Since only two protocols, TCP and UDP, need to specify port numbers, we select up to 11 additional bits from the port numbers for these protocols. Since a port number is bi-directional, i.e. either source or destination, and specified by a range with upper and lower bounds, we select one of the port field by an additional bit to denote the direction and then select additional 10 LSBs or 6 MSBs from the port field by using the *precision directed grouping*, which we will describe in Section 2.4.2. Typically, a server port (dense area) designates a specific port number between 0 and 1023 while a client port (sparse area) uses a random port numbered from 1024 to 65536. Thus, lower 10 bits are used for a server port while upper 6 bits are used for a client port. Thus, a hash key is concatenated from [protocol field], [direction bit] and [10 LSBs | 6 MSBs in one of the port field]. Since a server port has a higher partitioning efficiency than a client port, we use the server port regardless of direction if a rule specifies a server port. If a rule specifies client ports in both source and destination ports, we use the 6 MSBs and spread the rules in both source and destination hash tables. This rule spreading is described in detail in Section 2.4.2.

2.3.2 The Second-Level Partitioning

The second level partitioning only applies to buckets larger than the threshold after the first level partitioning. Since the second-level hash key must be disjoint from the first-level hash key, we only consider source and destination IP addresses. To limit the size of the hash table, we only select a subset of the 64-bit fields as a second level hash key. In our implementation, we use a 16-bit hash key. We use the following four different hash key selection algorithms.

1. *MSB pattern* (represented as *MSB*): With this criterion, a 16-bit hash key is made by concatenating 8 MSBs from source and destination IP address fields. The idea is that most of prefix mask selects the first few significant bits from an IP address field. The time complexity of this key selection algorithm is *O(1)*.
2. *Exponential growing pattern* (*Exp*): With this criterion, a 12-bit hash key can be made by selecting the bit position corresponding to the exponential function of 2, namely $b_1b_2b_4b_8b_{16}b_{32}$ from both source and destination IP addresses. The idea is that the lower the position of a bit in an IP address field, the more likely to be masked out. We add extra two bits b_6b_{11} to create a 16-bit hash key. The time complexity of this key selection algorithm is also *O(1)*.
3. *Mask distribution pattern* (*Mask*): The basic idea of this heuristic method is that don’t care bits in a classification space do not provide any information. Thus, each bit b_i in the classification space has the information in inverse proportion to the number of don’t care bits in the bit position of rule definitions. The procedure of finding this key is as follows. For each bit position b_i we sum the number of non-don’t care bits in all the rules in a rulebase, and accumulate them from the MSB to the LSB. For a *k*-bit hash key, we select a bit if the accumulated value of

the bit position is the multiple of the total accumulated value divided by k. The time complexity of this key selection algorithm is $O(kn)$ assuming that the total number of rules in a rulebase is n. In our experimentation, k is 16.

4. *Entropy-maximizing pattern* (*Ent*): To find a good hash key we use the notion of *entropy*, which is used in information theory [11]. As known widely, the entropy is maximized when all the entries have the same probability of occurrence. Thus, we can find a good hash key through the calculation of entropy. Using the notion of entropy, a hash key $\boldsymbol{K_\sigma}$ of length σ can be expressed recursively by $\mathbf{K_\sigma} = \mathbf{K_{\sigma-1}} \oplus \mathbf{q}$, where $\oplus$ is the concatenation operator and q is the bit from the classification space that produces the maximum entropy. The algorithm starts by calculating the entropy for the hash key of length 1 and determines the bit position that produces the maximum entropy value. Then, the algorithm repeats this process for the hash key of length 2 and so on until the length of the hash key reaches σ or the entropy does not increase further. Based on this algorithm, we create a 16-bit hash key by selecting an 8-bit hash key from each IP address field. The time complexity this algorithm is $O(n \cdot [\frac{s}{2}(2w - s + 1)])$, where w is the length of classification space, s is the length of a hash key, and the n is the total number of rules in a rulebase. The detailed discussion of the entropy-maximizing key selection algorithm can be found in [16].

2.4 Adaptation of the Algorithm for Prefix Mask and Range Specification

So far our discussion implicitly assumed exact value matching for packet classification. However, a rule definition often includes field descriptions with prefix mask or range specification. In this section, we will discuss how our proposed algorithm can handle these different field specifications.

Table 3. A rulebase and its hash tables with two different hash keys of length 2.

Rule	**Field Description ($b_0b_1b_2b_3b_4b_5b_6b_7$)**
R0	0000 0000
R1	0110 0000
R2	1000 0000
R3	1*10 0000

Index	**Hash Key b_0b_1**	**Hash Key b_0b_2**
00	**R0**	**R0**
01	**R1**	**R1**
10	**R2, *R3***	**R2**
11	***R3***	**R3**

2.4.1 Prefix Mask Field

This is commonly used to specify the range of an IP address field. Table 3 shows a rulebase example with prefix masks and its two different hash tables. The issue here is that we may need to duplicate a rule into multiple entries in a hash table if the rule contains a field specification with masks.

With b_0b_1 as a hash key, ***R3*** needs to be spread over two entries indexed by 10 and 11 since b_1 is don't care term in ***R3***. We call this issue *rule spreading*, which may increase the size of the result hash table by duplicating rules. However, by selecting b_0b_2 as a hash key, the rule spreading can be avoided as. To avoid this rule spreading as much as possible, we need to modify the entropy-maximizing key selection algorithm such that when calculating the entropy of a bit, the algorithm must ignore a rule whose definition specifies don't care condition for the selected bit. This is simply because don't care bits do not add any information to the system in terms of entropy.

2.4.2 Range Field

This is commonly used to specify a TCP or a UDP port description. As described in Section 2.3.1, typically a server port designates a specific port number between 0 and 1023 while a client port uses a random port number between 1024 and 65536. The basic idea here is to transform the range specification to exact or prefix mask specification. We can use *range to prefix conversion* [5], which splits a given arbitrary range to a group of prefix masks. For example, a 16-bit range [1024, 65535] can be split to six prefix masks such as 000001*, 00001*, 0001*, 001*, 01*, 1*. However, this method results in extensive rule spreading in our algorithm, which is not desirable. Alternatively, we propose *precision-directed grouping* in this paper.

It is straightforward to split a rule with a range specification into multiple rules with exact values. For example, a rule with a range [71, 74] can be split into four rules with exact values from 71 to 74. However, a rule with a wide range such as [49152, 65535] can create a huge number of rules (16,384). Fortunately, a TCP/UDP port description with a range is usually biased. For example, 80% of port numbers used in most rules are under 3,999 although the total number of port reserved is much higher (0 ~ 49,151) [13]. Thus, we can group a different number of rules depending on the density of the range. For the 16-bit port range, we use the 10 LSBs as a hash key for dense area [0, 1023], creating a single entry per port, while we use the 6 MSBs as a hash key for sparse area [1024, 65535], creating 63 entries, i.e. 1024 ports per entry.

3 Experimentation and Results

In this section, we demonstrate the performance of the proposed algorithms for 5-dimensional classification. Since it is difficult to obtain large real-life classification rulebases, we synthesized large rulebases from real-life packet traces. The packet traces were collected from PUBNET for five to eight hours during three days, 7/24/01, 12/14/01 and 12/17/01, respectively [14]. One-hour trace is nearly 70 million packets. The first trace is used to synthesize the rulebase and others are used as data for packet classification. To create a synthetic rulebase that resembles real-life rulebases, we carefully synthesized a rulebase by following the rulebase characteristics observed from real-life firewall applications [1, 2]. The detailed guidelines are described in [16]. All of our experimentation was performed in 1.7GHz Pentium IV system with 512MB of memory running Linux.

3.1 Rulebase Partitioning

3.1.1 First Level Partitioning

Figure 1 shows the results of the first level partitioning by displaying the average and maximum size of a sub-rulebase after the partitioning. By the partitioning we can reduce the average size of a rulebase substantially. For rulebases with 10K, 100K, and 500K rules, the reduction ratios are 0.0029, 0.0014, and 0.0014 respectively. This is very significant since we can reduce the size of a rulebase by more than two orders of magnitude by a single memory lookup to the corresponding hash table. However, as you can see from the maximum size of a sub-rulebase in the figure, rules are not evenly distributed in the partitioned rulebases. The largest sub-rulebase contains about 24% of rules of the original rulebase in all the rulebases tested. As we can predict, these rules are related to HTTP service, which corresponds to protocol 6 and port 80. The numbers of sub-rulebases over the threshold (16 rules per sub-rulebase) are 141, 192, and 1114 for 10K, 100K, and 500K cases. For these sub-rulebases we perform the second-level partitioning. All of the first level partitioning is completed in less than one second in our experimentation platform. As a side effect of the first level partitioning, we observe that the partitioning more than doubles the total number of rules due to rule spreading. The actual inflation ratio is 2.42.

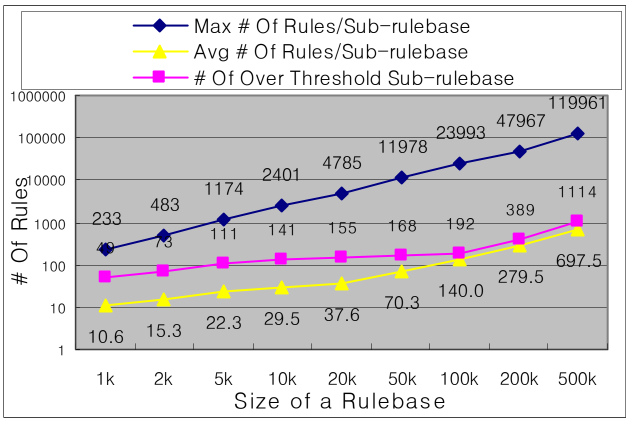

Fig. 1. The result of the first level partitioning

3.1.2 Second Level Partitioning

Figures 2 and 3 show the results of the second level partitioning with various hash key selection algorithms. Figure 2 shows the average number of rules per sub-rulebase while the Figure 3 shows the size of the largest rulebase. Assuming the entropy-maximizing key selection, the second-level partitioning further reduces the sub-rulebase by reduction ratios of 0.054, 0.054, and 0.052 for 10K, 100K, and 500K rulebases. When the first-level and the second-level partitioning are combined, 10K, 100K, and 500K rulebases are reduced to 1.6, 7.6, and 36.6 rules per sub-rulebase on average, which corresponds to reduction ratios of 0.00016, 0.000076, and 0.000073. This is very significant since we can reduce the size of a rulebase by more than four orders of magnitude by just two memory lookups to the hash tables.

The second level partitioning is also very effective in reducing the largest sub-rulebase, which contained 24% of the entire original rulebase after the first level partitioning. Assuming the entropy-maximizing key selection, with the second level partitioning we can reduce the size of the largest rulebase to contain 31, 258, and 1281 rules in 10K, 100K, and 500K rulebases respectively. This suggests that for a 100K rulebase we only need to compare a packet to those 258 rules in the worst case during classification phase. Note that the second-level partitioning is more effective than the first-level partitioning in reducing the largest rulebase. This is expected since the first-level partitioning partitions the original rulebase according to the Internet services. Thus, the largest sub-rulebase governing the HTTP service is grouped into a single sub-rulebase. On the contrary, the second-level partitioning can reduce this largest sub-rulebase substantially by considering the distribution of rules in the classification space with the entropy-maximizing key selection algorithm.

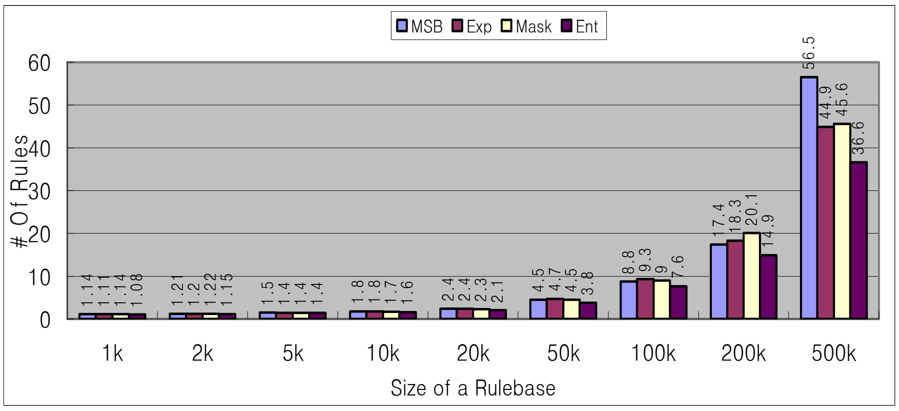

Fig. 2. The average size of a sub-rulebase with different key selection algorithms

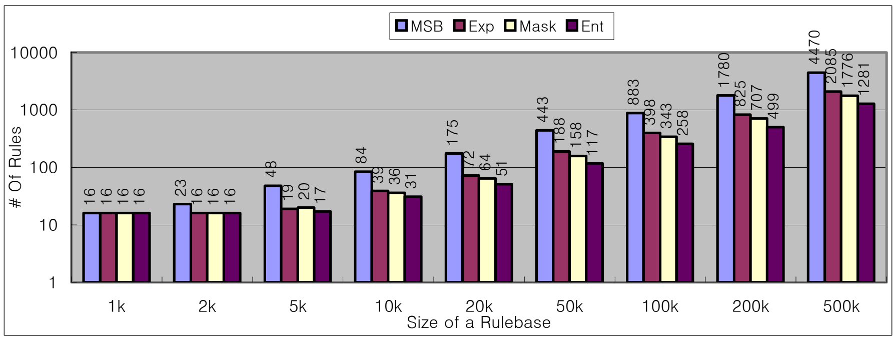

Fig. 3. The maximum size of a sub-rulebase with different key selection algorithms

Figures 2 and 3 also compare the effectiveness of different key selection algorithms. As expected, the entropy-maximizing key selection gives the best result. In particular, it can reduce the size of the largest sub-rulebase substantially compared to other heuristics. For 10K, 100K, and 500K rulebases, the entropy-maximizing key selection is more effective compared to MSB pattern key selection by factors of 2.38, 3.42, and 3.49. All the key selection algorithms show comparable results on the average size of a sub-rulebase. However, the entropy-maximizing key selection is the most effective in all the cases, especially for rulebases larger than 100K rules.

It is hard to judge the scalability of our rulebase partitioning from Figures 2 and 3 since the size of a rulebase does not increase linearly in the horizontal axis. But, we carefully scrutinized the numbers and could verify that both the maximum and the average size of a sub-rulebase grow linearly as we increase the size of rulebase. Note that these results assumed only two-levels of partitioning. If necessary, more levels of partitioning can be employed to significantly reduce the size of sub-rulebases. Likewise, we studied the memory requirement of our algorithm in terms of the growing rates of the total number of rules, the total number of buckets, and the total number of buckets over threshold [16]. We omit them here due to space limitation, but we find that all the numbers show a good scalability.

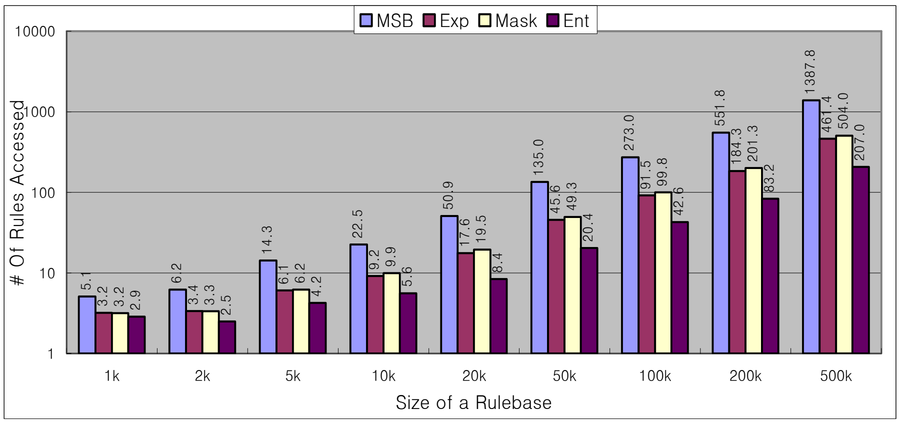

Fig. 4. The packet classification results

3.2 Classification Performance

To demonstrate the classification performance of our algorithm, we use real-life packet traces collected from PUBNET. The size of packet trace is 202 million packets, which is big enough to obtain the confidence of our experiment. After the first and second-level partitioning, we assume the worst-case search algorithm, i.e. a simple linear search, to find the matching rule in the final sub-rulebase so that we can show the pure effectiveness of the proposed partitioning algorithm.

Figure 4 shows the classification performance in terms of the average number of rules accessed to find a matching rule. Figure 4 reveals that our algorithm accesses only 5.6, 42.64, and 207.02 rules for 10K, 100K and 500K rulebases respectively. This is very promising since the best-known algorithm [1] requires at least 13 memory accesses for 5K rules. On the other hand, our partitioning algorithm needs to access only 5.6 rules in addition to 2 memory accesses required for hash table lookups although the worst-case linear search is used in the final sub-rulebase. By employing a more efficient algorithm [2, 5, 12] to find a matching rule in the final sub-rulebase, we can further improve the classification performance.

4 Conclusion

The growth of Internet in applications and protocols makes the size of a rulebase rapidly increasing. However, most of existing works mainly focus on relatively small classifiers, e.g., with less than 20K rules. Beyond this size, most of existing schemes may not scale either due to the memory explosion or due to the slowdown of classification. To address this issue, we propose a new classification algorithm that achieves the scalability by hierarchically partitioning a rulebase into many smaller independent sub-rulebases. By using the same hash key used in the partitioning a classifier can inspect an incoming packet and find its relevant sub-rulebase with a few memory lookups to the hash tables. Thus, the classification performance mainly depends on how much the final sub-rulebase can be reduced compared to the original rulebase. To make the reduction effective, we use the notion of entropy that finds a good hash key considering the distribution of rules in the classification space.

With synthesized rulebases of sizes ranging from 1K to 500K rules, we evaluate the effectiveness of the algorithm in both partitioning and classification. The experimental results show that two-levels of partitioning can substantially reduce the size of a rulebase. As a result, during classification a classifier needs to access only 4.2, 20.4, and 207 rules on average for rulebases with 5K, 50K, and 500K rules. This is very promising since one of the best-known algorithms [1] requires at least 13 memory accesses for 5K rules. Furthermore, we show that the proposed algorithm has the unique scalability both in space and in time as we increase the size of rulebase.

References

1. P. Gupta and N. McKeown, "*Packet Classification on Multiple Fields*", In Proceedings of the ACM SIGCOM '99, Vol. 29, issue 4, August 1999.
2. F. Baboescu and G. Varghese, "*Scalable Packet Classification*", In Proceedings of the ACM SIGCOM '01, Vol. 31, August 2001.
3. *Flow Analysis of Passive Measurement Data*, http://pma.nlanr.net/PMA/Datacube.html.
4. T. V. Lakshman and D. Stiladis, "*High-speed Policy-based Packet Forwarding using Efficient Multi-dimensional Range Matching*", In Proceedings of the ACM SIGCOMM '98, Vol. 28, pp. 191-202, 1998.
5. V. Srinivasan, S. Suri, G. Varghese, and M. Valdvogel, "*Fast and Scalable Layer Four Switching*", In Proceedings of the ACM SIGCOMM '98, Vol. 28, pp. 203-214, 1998.
6. V. Srinivasan, G. Varghese, and S. Suri, "*Packet Classification Using Tuple Space Search*", In Proceedings of the ACM SIGCOM '99, Vol. 29, pp. 135-146, August 1999.
7. M. M. Buddhikot, S. Suri, and M. Waldvogel, "*Space Decomposition Techniques for Fast Layer-4 Switching*", In Proceedings of the IFIP Sixth International Workshop on Protocols for High Speed Networks. Vol. 66, No. 6, pp. 277-283, August 1999.
8. A. Feldmann and S. Muthukrishnan, "*Tradeoffs for Packet Classification*". In Gigabit Networking Workshop of the Proceedings of the IEEE INFOCOM '00. March 2000.
9. T. Woo, "*A Modular Approach to Packet Classification: Algorithms and Results*", In Proceedings of the IEEE INFOCOM '00. March 2000.
10. P. Gupta and N. McKeown, "*Packet Classification using Hierarchical Intelligent Cuttings*", In Proceedings of the Hot Interconnects VII, 1999.
11. Robert B. Ash, "*Information Theory*", Dover Publications, 1st edition, November 1990.
12. H. Kim, J. Heo, L. Choi, and S. Kim, "*Taming Large Classifiers with Rule Reference Locality*", In Proceedings of the ICOIN. Vol. 1 pp. 35-50, February 2003.
13. *IANA Port Number Assignment*, http://www.iana.org/assignments/port-numbers, 2002.
14. Korea Network Information Center, http://www.nic.or.kr/.
15. Elizabeth D. Zwicky et al. "*Building Internet Firewall*", 2nd edition. O'Reilly, 2000.
16. Jaesung Heo, "*Scalable Packet Classification through Maximum Entropy Hashing*", M.S. thesis, Department of Electronics and Computer Engineering, Korea University, 2003.