Packet Classification with Network Traffic Statistics

Yaxuan Qi and Jun Li
Research Institute of Information Technology (RIIT), Tsinghua University
Beijing, China, 100084

***Abstract*-- Packet classification on multiple header fields is one of the basic techniques for policy enforcement applications in network devices. In this paper, we analyzed the existing packet classification algorithms from a theoretical point of view, focusing on the information exploitation in each algorithm. A novel packet classification scheme is proposed that exploits not only the structural characteristics of network packet classifiers but also the statistical characteristics of network traffic flows. The proposed algorithm D-Cuts (Dynamic Cuttings) is based on a decision tree data structure similar to but improved than the previously well-known algorithm HiCuts. However, the memory allocation function in D-Cuts is significantly improved by introducing the statistical characteristics of network traffic. A set of unified discrimination criterion based on entropy measurement is also adopted by D-Cuts. Experimental results show that D-Cuts performs superior to other popular multiple-field packet classification algorithms. Its average search time is on the same order as for the HiCuts optimized for speed while its memory usage is on the same order as for HiCuts optimized for space.**

***Index terms*—packet classification, traffic statistics, search space.**

A. Introduction

The Internet is becoming a more and more complex place to live in because of its use for more and more mission critical tasks executed by organizations. It is desired that those critical activities not be subverted either by heavy traffic sent by other organizations or by malicious intruders. Traffic engineering, access control, and many other services require a discrimination of packets based on multiple fields of packet headers, which is called packet classification.

Packet classification is employed by policy enforcing devices to implement a number of advanced Internet services, such as policy-based routing, access-control, service differentiation, load balancing, traffic shaping and traffic billing. Each service requires the Internet devices to classify packets into different flows and then perform appropriate actions depending upon which flow the packet belongs to. These flows are specified by a classifier containing a set of rules.

With the rapid development of policy enforced networks, packet classification becomes more and more important and there is a need for efficient packet classification algorithms to enable high speed policy enforcement. Several popular algorithms have been proposed and some of them provide reasonable solutions. Most of the new algorithms proposed in recent years, especially the heuristic algorithms, are inspired by the observation of redundancy in the data structure of real-life classifiers. Such observation is introduced into the design of packet classification algorithms to improve the classification performance. The importance of the observation against real-life classifiers is undeniable, but it was a bottom-up approach and hence lack of theoretical guidance. Therefore, it is hard to further improve and refine those algorithms following this direction.

Our research starts from the theoretical analysis on the packet classification problem. We compared and analyzed the existing algorithms from two aspects: the exploitation of the heuristic information and the division of the search space. Based on the conclusion of such analysis, we present a novel packet classification scheme that adopts network traffic statistical characteristics into the decision tree based classification algorithms. Compared to the well-known algorithm HiCuts, its average search time is on the same order as for the HiCuts optimized for speed while its memory storage requirements are on the same order as for HiCuts optimized for space.

The rest of the paper is organized as follows. Section B gives a mathematic definition of the packet classification problem; Section C compares and analyzes the most popular existing packet classification algorithms; Section D presents the reasoning to introduce network traffic statistical characteristics into the decision tree based data structure; Section E describes the proposed algorithm Dynamic Cuttings (D-Cuts); Section F illustrates the experimental results of D-Cuts in comparison with the existing algorithms; as a summary, Section G states our conclusions.

B. Mathematic Definition Of Packet Classification

Generic packet classification classifies a packet based on multiple fields of its header. Each rule of the classifier specifies a *class* that a packet may belong to, based on certain specifications on the F fields (dimensions) of the packet header. The *class* uniquely determines the action associated to the rule. Each rule has F components. The ith component of rule R, referred to as $R[i]$, is a regular expression on the ith field of the packet header. A packet P is said to *match* a particular rule R, if, the ith field of the header of P satisfies the regular expression $R[i]$, for all $0 \le i < F$.

The *classes* specified by the rule set may be overlapping, i.e. one packet can match several rules.

From a theoretical point of view, the F fields of the packet header make up a multi-dimensional space, which is called the *search space* in this paper. Each of the F fields is a dimension of the search space. A packet P is a point in the multi-dimensional search space. For the generalized range matching, the regular expression $R[i]$ refers to a range in the ith dimension of the search space and all of these ranges make up a *F*-dimensional hyper-cube. If a packet P matches a particular rule R, the point P falls into the hyper-cube specified by R. Therefore, packet classification can be treated as a point location problem in computing geometry.

The point location problem is inherently hard to solve. It has been proved [2] that in its fullest generality, packet classification requires either $O(\log^{F-1} N)$ time and $O(N)$ space, or $O(\log N)$ time and $O(N^F)$ space where N is the number of rules, and F is the number of header fields. Therefore, it is relatively simple to perform packet classification at high speed using large amounts of storage, or at low speed with small amounts of storage. When matching multiple fields (in another word, searching in multiple dimensions) simultaneously, however, it is difficult to achieve both high classification speed and modest storage in the worst-case.

C. Analysis Of Previous work

The design of classification algorithms is encumbered by worst-case bounds on search time and memory requirements that are so onerous as to make brutal force algorithms unusable [3]. Therefore, it will be futile to try to find a global optimized algorithm under all circumstances. Instead we must search for structures or characteristics of certain classification problems that can be exploited in pursuit of algorithms that are "fast enough" and use "not too much" space.

The simplest classification scheme is a linear search of each rule of a classifier. While it is very efficient in terms of storage requirements and update time, linear search requires a comparatively long search time of $O(N)$. This makes it impractical to deal with large size classifiers as the search time increase linearly with larger N.

In resent years, a variety of packet classification schemes have been proposed to solve the general problem of multi-dimensional packet classification. The generic idea to deal with large classifiers is *divide and conquer*: Most of the existing algorithms appropriately divide the multi-dimensional search space into a certain number of sub-spaces. Since there are fewer rules in each sub-space, the original classification problem is simplified and hence easier to deal with. Each algorithm has its own way to divide the search space. Different ways in which to apply the division depend on the information exploited by the algorithms. Such information includes:

1. Basic attributes of the search space, such as its dimensionality and the ranges of value in each dimension.
2. Structural properties of real-life classifiers, including the distribution, redundancy and many other characteristics of the given rule set.
3. Statistical characteristics of real-life (or specific, particular) networks, such as the traffic flow statistics and the hit rates of the rules.

The above items 1 and 2 are static information. They are definitely specified by certain classification problems (IDS/IPS interacting with firewall and introducing firewall policies reactively can be treated as a special case and does not impact the generalness of this claim). The item 3 is dynamic information because statistical characteristics of certain networks are time-variant.

Srinivasan et al. [5] proposed two algorithms Grid-of-trie and Cross-producting. Like Hierarchical tries [4] and Set-pruning tries [4], Grid-of-trie uses a trie-based data structure and the search space is equally divided bit by bit at each trie node. Superior to other trie-based algorithms, Grid-of-trie adopts *switch pointers* to avoid the time-consuming back tracking search. These switch pointers allow the query process to switch from one sub-space to another. Cross-producting divides the search space according to the rule segmentations along each dimension. Each segment refers to a sub-region in one of the *F* dimensions, and the cross-product of the *F* sub-regions makes up of a sub-space. Search can be done quickly by separate lookups on each dimension and then indexing into the cross-product table.

Gupta and McKeown [1,3] introduced two new algorithms, RFC and HiCuts. HiCuts is based on a decision tree structure. At each tree node, the current search space is equally cut (divided) along a chosen dimension. Which dimension to cut and the number of cuttings (divisions) depend on the characteristics of the rules belong to the node. Different from trie-based algorithms, the division in HiCuts is not a simple binary-division. The algorithm determines the number of divisions according to the given rule set. RFC is really an improved form of Cross-producting that significantly compresses the cross-product table. In the first phase of RFC, the division of the search space is the same as that of Cross-producting. However, RFC is a recursive algorithm and does not obtain the classification result though one table lookup. The main idea of RFC is to place the smaller cross-products into equivalence classes

before combing them to form larger cross-products. This equivalencing of partial cross-products considerably reduces memory requirements, because several original cross-product terms map into the same equivalence class [8].

Objective evaluation of the performance of each algorithm can be done both in worst-case analysis and experimental comparison. We list the worst-case complexity of time and space in **Table.1**, from which we found that it is not obvious that some algorithms are superior to others. However, experimental results (given in SECTION F) show that the decision tree based algorithm mostly works well. This is because the flexible data structure of the decision tree makes it possible to exploit more useful information of real-life classifiers into the design of packet classifiers. We summarize the existing algorithms as three aspects:

1. The algorithm generality. Some of the existing algorithms, such as Grid-of-trie, are designed for 2-dimensional packet classification and not efficient for multi-dimensional problems. Although Cross-producting works well with small number of rules in multiple fields, it is not able to handle large classifiers. RFC is very efficient for multi-dimensional classification, but it is found that RFC consumes too much storage for classifiers with more than 2000 rules. HiCuts mostly works well with thousands of multi-dimensional rules. While it is slower compared to RFC, HiCuts is able to give a practical solution for large classifiers. This is because the memory requirement for the decision tree data structure does not explodingly increase.
2. The information exploitation. Grid-of-trie exploits the information that some of the rules share the same part of the prefix, so the searching process can switch from one sub-space to another in order to avoid the redundant back-tracking search. RFC takes advantage of the observation that, in real-life classifiers, the number of overlapping regions is considerably smaller than the worst-case, thus a recursive mapping can be applied. HiCuts exploits much more heuristic information in such structural redundancy, especially some abstract information, such as the discriminating ability in each dimension. More information exploited, more efficient will the algorithms be.
3. The search space division. There are two ways to divide the search space: dividing equally and dividing according to the rules. Grid-of-trie and HiCuts adopt the former way. At each node, Grid-of-trie bisects the search space while HiCuts divides (cuts) the search space into a certain number of equal-sized sub-spaces. Cross-producting and RFC use the latter method, dividing each dimension of the search space into segments according to the given rules. Searching in equal-sized space needs just one operation while in unequal-sized space (division according to the rules) it requires a binary search with $\log(N)$ operations. On the other hand, equal-sized division is not as discriminative as the division according to the rules, i.e. it requires larger number of divisions to discriminate the give rules. **Figure.1** is a simple example to compare the two ways for division. Although at each node, HiCuts applies an equal-sized division, but when considered globally, the whole decision tree virtually divides the search space into unequal-sized sub-spaces.

Table.1 Comparison of popular algorithms. In this table N is the number of rules, W is bit-width of header vector in a certain dimension (e.g. for IP address, $W = 32$), F is dimensionality of the search space.

Algorithm	Worst time	Worst space
Linear Search	$O(N)$	$O(N)$
Hierarchical Tries	$O(W^F)$	$O(N)$
Cross-producting	$O(FW)$	$O(N^F)$
Grid-of-tries	$O(W^{F-1})$	$O(N)$
RFC	$O(F)$	$O(N^F)$
HiCuts		

As a conclusion, the reason why HiCuts can exploit more useful information is that the decision tree based data structure of HiCuts is very flexible to adapt to different classifiers. The proposed algorithm D-Cuts (Dynamic Cuttings) is also based on decision tree data structure. Different from HiCuts, D-Cuts exploits not only the static information, but also the dynamic information of certain networks. In addition, D-Cuts uses an improved discrimination criterion in building the decision tree.

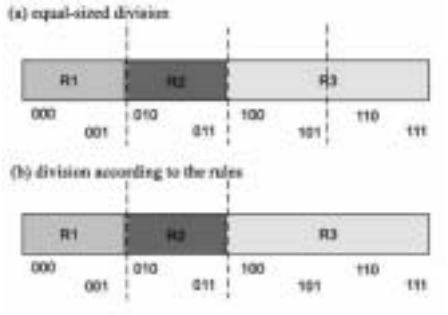

Figure.1 Two ways of the search space division in one dimension for three rules ($R1, R2, R3$). (a) refers to equal-sized division. It needs 4 divisions to discriminate the 3 rules and 1 operation to search. (b) is the division according to the rules. It needs 3 divisions to discriminate the rules but more (up to 2) search operations.

D. Statistical Characteristic Of Network Traffic

Most of the existing classification algorithms exploit the static information, such as the structural characteristics of real-life classifiers. They assume all incoming packets are distributed uniformly in the search space. However, it is unlikely that the traffic in a certain network evenly spread over all IP addresses and/or port numbers. For example, most Internet sessions are usually Web applications so that majority packet headers are having destination port numbers of 80 or 443 (HTTP or HTTPS). In a particular network, some of the rules may be set to prevent networks from virus or other attacks and thus applicable for almost no traffic being classified. Although most of time there is no packet matches them, these rules have to be kept in the classifier for the sake of security, and thus make the division of the search space more complex.

Each network has its own traffic patterns, and the packet classification process is affected by the dynamic characteristics to a certain extend. Our study focuses on the exploitation of network traffic characteristics to achieve more time-efficient and less memory-consuming classification algorithms. Challenges of adopting this new idea include:

1. Extracting the dynamic information of network traffic and expressing them in an appropriate form that can be employed in classification algorithms. The dynamic characteristics of network traffic exploited in our algorithm are the IP address and port number statistical distribution in the search space. Periodic sampling is a practical way to gather statistical information of IP and port distribution characteristics. After normalization, the statistical information is introduced into the proposed classification algorithm as prior probability of the IP addresses and port numbers.
2. Optimizing the algorithm dependency to the statistical characteristics of network traffic. Different from the static structural characteristics of given classifiers, the dynamic characteristics of network traffic are time-variant. In the proposed D-Cuts algorithm, the extent to which it depends on the traffic statistics is determined by setting two bounds for the space factor.
3. Applying the statistical characteristics in classification algorithm. D-Cuts exploits statistical characteristics of network traffic by dynamically adjusting the memory allocation function. Nodes corresponding to larger traffic flow will be allocated with more memory storage to reduce the depth of the sub-trees.

Details of the considerations will be elaborated in the next section.

E. The Proposed Algorithm D-Cuts

E.1 An Example

To illustrate the D-Cuts algorithm we proposed in this paper, a simple 2-dimensional classifier (similar to the classifier used in [4]) is used for demonstration, shown in **Table.2**. This classifier is represented with two fields in a 2-dimensional space (*X*-*Y* plane), where each rule is represented by a rectangle (**Figure.2**).

Like HiCuts, the classification scheme D-Cuts is based on a decision tree structure. However, the way to implement cuttings on the search space is improved by exploiting not only the structural characteristics of the specific classifier, but also the network traffic statistical characteristics. Spaces where there is more traffic will be cut into more sub-spaces in order to reduce the depth of the sub-trees.

Table 2. A sample classifier with 6 two-dimensional rules, the search space is X-Y plane.

Rule	Field1(*X*)	Field2(*Y*)
R1	00*	00*
R2	0*	01*
R3	1*	0*
R4	01*	00 *
R5	0*	1*
R6	1*	1*

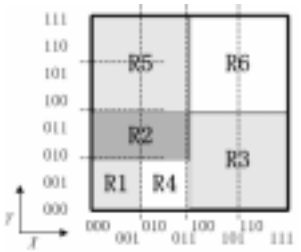

Figure.2 Geometric representation of the sample classifier in **Table.2**.

In HiCuts, space available for each *N*-size node (node with N rules) v is determined by the function $SpaceAv(v)$:

$$SpaceAv(v) = N * spfac \quad (1)$$

Where *spfac* is the *space factor* [1] and often takes value through 1 to 4 for a time/space tradeoff. A larger *spfac* is likely to consume more memory while cost less search time. $SpaceAv(v)$ sets an upper bound for the number of cuttings and, in the example, we simply define the cutting number $numCuts(v)$ by:

$$numCuts(v) = SpaceAv(v) \quad (2)$$

Figure.3 shows how HiCuts works for the sample classifier listed in **Table.2**.

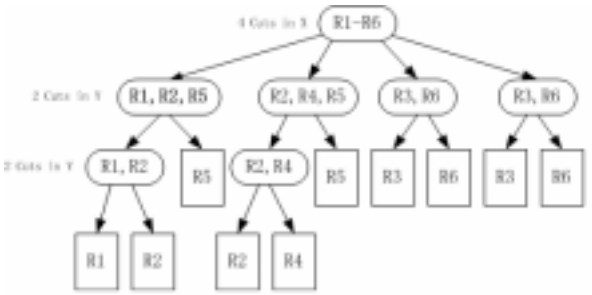

Figure.3 Decision tree built by HiCuts for the sample classifier listed in **Table.2**. Leaf nodes (rectangle) have only one rule while internal nodes (ellipse) have more than one rules.

In D-Cuts, we simply revised the space allocation function for node *v* by:

$$SpaceAv(v) = N * P(v) * 10 \quad (3)$$

Where $P(v)$ is defined as the probability of traffic falling into the region covered by node v along X dimension. In case of 40% traffic falls into region [000, 001] of X dimension and the other 60% traffic distributed uniformly in [010, 111] along X, **Figure.4** illustrates the decision tree built by D-Cuts.

Figure.4 Decision tree built by D-Cuts for the sample classifier listed in **Table.2**, with 40% traffic falls into region [000, 001] of X dimension.

It can be derived from **Figure.3** and **Figure.4** that the number of nodes in D-Cuts does not increase while the *average* depth of decision tree decreases, comparing to HiCuts. Worst-case search time (determined by the depth of the decision tree) stays the same while the average search time is improved because 40% traffic travels through the 2-level sub-tree in D-Cuts in stead of the 3-level sub-tree in HiCuts.

E.2 Extracting Statistical Characteristics of Network Traffic

Network traffic statistical characteristics are obtained by periodic sampling. First, we get the packet header distribution information along each dimension, such as IP prefixes (16-bit for *B class*) and port number ranges. For example, a two-dimensional array of 4*65536 entries $\{count[d][i], d = 0,...,3, i = 0,...,65535\}$ can be used to count source/destination *B-class* IP prefixes (for *d*=0 and 1) and source/destination port numbers (for *d*=2 and 3), e.g. $count[3][80]$ is the number of sampled packets with destination port 80.

After sampling step, a normalization process is applied to $\{count[d][i]\}$. We use $\{prior[d][i], d = 0,...,3, i = 0,...,65535\}$ to denote the normalized distribution of network traffic, where

$$prior[d][i] = \frac{count[d][i]}{\sum_{j=0}^{65535} count[d][j]} \quad (4)$$

E.3 Introducing Statistical Characteristics Of Network Traffic To D-Cuts

SECTION **E.1** gives a simple version of function $SpaceAv(v)$ to illustrate the exploitation of network traffic statistical characteristics. Here we define two general forms of function $SpaceAv(v)$.

FORM-1:

$$\begin{aligned} SpaceAv(v) &= N * spfac(v) \\ &= N * (spfac_{min} + (P_v^l - \min(P^l)) * K) \end{aligned} \quad (5)$$

$$P_v^l = \frac{1}{4}\sum_{d=0}^{3}\sum_{i=a_d}^{b_d} prior[d][i] \quad (6)$$

$$\text{for } \max(P^l) - \min(P^l) > 0$$
$$K = (spfac_{max} - spfac_{min}) / (\max(P^l) - \min(P^l)) \quad (7)$$
$$\text{for } \max(P^l) - \min(P^l) = 0$$
$$K = 0$$

where N is the number of rules associated with node *v*; P_v^l is the prior probability assigned to node *v* at level *l* of the decision tree; $[a_d, b_d]$ is the search range of node *v* along the *dth* dimension; $\max(P^l)$ and $\min(P^l)$ are the maximum/minimum prior probability among all nodes in the same level of the decision tree; $spfac_{min}$ and $spfac_{max}$ are bounds of the space factor.

FORM-2:

$$SpaceAv(v) = N * spfac(v)$$
$$= N * BOUND(P_v^l * D^l * spfac_{avg}, spfac_{min}, spfac_{max}) \quad (8)$$

where $spfac_{avg} = (spfac_{max} + spfac_{min})/2$; D^l is the number of nodes at the same level as node *v*; P_v^l takes the same definition in (6); and $BOUND(a,b,c)$ is defined as the following that has value *a* with lower bound as *b* and upper bound as *c* (for $b \le c$):

$$BOUND(a,b,c) = \begin{cases} a & b \le a \le c \\ b & a < b \\ c & a > c \end{cases} \quad (9)$$

Different from the space factor $spfac$ in HiCuts, the space factor *spfac* is not a constant in D-Cuts. It takes value in $[spfac_{min}, spfac_{max}]$. Here $spfac_{min}$ can be specified for space-optimization, while $spfac_{max}$ is specified for time-optimization. Therefore, it can be seen in both FORM-1 and FORM-2 that nodes with larger P_v^l, i.e. larger traffic volume, will have more memory reserved for cuttings to increase the search speed.

E.4 Selecting The Number Of Cuttings $numCuts(v,d)$

An approximate memory measurement for the cutting of node *v* is defined as [1]

$$sm(v,d) = \sum_{i=1}^{numCuts(v,d)} numRules(v_i) + numCuts(v,d) \quad (10)$$

where *d* is the dimension to cut, v_i is a child node of *v*, $numRules(v_i)$ is the number of rules colliding with v_i (Rule *R* is said to "collide with" node *v* means *R* spans, cuts or is contained in the range associated with *v*). For $d = 0,...,3$ $numCuts(v,d)$ is determined by maximizing $sm(v, d)$ in (10) with the limit:

$$sm(v,d) < SpaceAv(v) \quad (11)$$

E.5 Selecting The Dimension To Cut $dimCut(v)$

To select a dimension to cut at node *v*, $dimCut(v)$ is determined by the discriminative functions that can be found in [1] and [7]. In our experiment, we found that under certain circumstances, these discriminative functions take the same value, i.e. they cannot determine which dimension is more discriminative to apply the cuttings. **Figure.5** is an example to show that the discriminative function (given in [1])

$$\min_{\forall d}(\max_{\forall i}(numRules(v_i, d))) \quad (12)$$

fails to determine $dimCut(v)$.

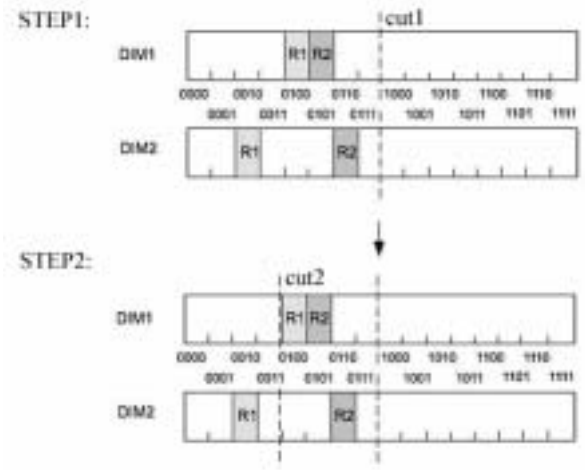

Figure.5 In STEP1, cut1 divide the search space [0000, 1111] into two equal-sized sub-spaces [0000, 0111] and [1000, 1111]. Both DIM1 and DIM2 have two rules fall into the sub-space [0000, 0111]. Therefore Eq.12 cannot discriminate which dimension to cut.

Note that most of the discriminative functions are based on the number of cuttings. We give an improved discriminative function that does not depend on the number of cuttings. This function describes the uniformity of the rule segments distributed in each dimension. For the equal-sized division at each tree node in D-Cuts, it is favorable to divide the dimension with most uniformly distributed segments. **Eq.13 and Eq.14** give the definition of the improved discriminative function

$$\max_{\forall d}(segEntropy(v,d) / numSeg(v,d)) \quad (12)$$

$$segEntropy(v,d) = \sum_{i=1}^{numSeg(v,d)} \frac{seg_{i,d}}{range_{v,d}} * \log \frac{range_{v,d}}{seg_{i,d}} \quad (13)$$

where $seg_{i,d}$ is the ith segmented rejoin along the dth dimension according to the rules belonging to node v; $range_{v,d}$ is the full range along dimension d in the search space of v; $numSeg(v,d)$ is the total number of segments along dimension d in the search space of node v. Adopting this discriminative function, the values computed for the two dimensions in **Figure.5** are: 0.43 and 0.54. Therefore it is more favorable to cut along the second dimension.

E.6 Implementation Flow-chart

As a summary, the flow-chart shown in **Figure.6** describes the implementation of D-Cuts. Because we use the Breadth First methodology to build up the tree, each time we try to build a node, all the nodes at the parent level is already done. This makes it possible to compute the two forms of $SpaceAv(v)$ for each node.

F. EXPERIMENTAL RESULTS

F.1 Databases

We evaluate D-Cuts both on real-life firewall and core router rule sets as well as on synthetic rule sets. The real-life rule sets are obtained from typical enterprise networks and major ISPs. The two firewall rule sets are named FW1, FW2, the two router rule sets are CR1, CR2, and the synthetic rule set is SN1. The number of rules in the five rule sets varies from 68 to 2000. All the classifiers are 4-dimensional with source/destination IP addresses represented as prefixes and source/destination port numbers represented as ranges. It is reported in [7] that the structural characteristics of firewall policy tables (rule sets) are different from core router access control list (ACL, also rule sets), e.g. most source port ranges in core routers are [0, 65535], while in firewall policy tables source port ranges are assigned more specifically. Statistical characteristics of network traffic are obtained by sampling or reasonable manual generation. Characteristics of the testing traffic flow are different from the sampling traffics in certain extent, in order to simulate the time-variant networks.

Figure. 6 Flow-chart for implementation of D-Cuts. *binth* in the chart is a constant parameter defined as the maximum number of rules for leaf nodes. In D-Cuts, we set *binth*=8.

F.2 Performance Evaluation

To test the performance of D-Cuts algorithm with both real-life and synthetic classifiers, we examined, for each classifier, the number of memory accesses in the query process (indicating search time) and the amount of memory usage for the whole data structure built by the algorithm. Note that the search time indicates the average search time which is the mean of the memory accesses of all testing packets.

The first comparison is done between HiCuts and D-Cuts on FW2. **Figure.7** shows the time/space performance in histograms for HiCuts-x ($spfac = x$) and D-Cuts (using two forms of $SpaceAv(v)$). It is obvious that in HiCuts, when the space factor *spfac* increases, the search time decreases while the memory usage blows up. Compared to D-Cuts, the search time is very close to the optimized search time (HiCuts-4) while the memory usage is just about 2/3 of HiCuts-4. D-Cuts effectively balances the storage space usage and the average searching time.

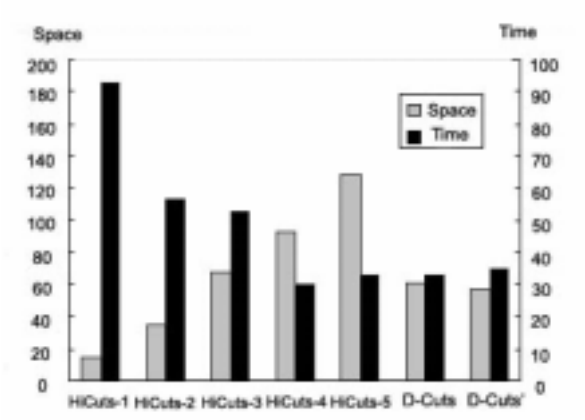

Figure.7 Time/Space trade-offs for FW2. Memory usage (Space) are shown by light histograms, Memory accesses (Time) are shown dark histograms. HiCuts-x refers to HiCuts with *spfac*=x. D-Cuts uses FORM1 $SpaceAv(v)$ function while D-Cuts' uses FORM2.

Figure.8 and **Figure.9** are the comparison on CR2 and SN1 via memory usage and average search time. It can be seen that the average search time of D-Cuts is on the same order as for the HiCuts optimized for speed (HiCuts-T_{opt} in the figures) while its memory usage is on the same order as for HiCuts optimized for space (HiCuts-S_{opt} in the figures).

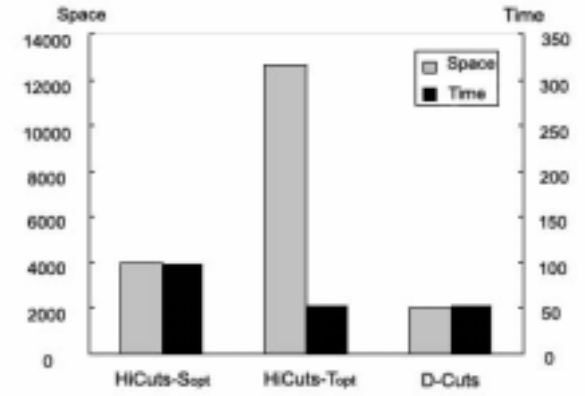

Figure.8 Time/Space trade-offs for CR2. Memory usage (Space) are shown by light histograms, Memory accesses (Time) are shown dark histograms. HiCuts-S_{opt} is space-optimized while HiCuts-T_{opt} is time-optimized.

Figure.10 is the comparison between HiCuts and D-Cuts via memory usage, average and worst-case search time on FW1. Note that the improvement of the average search time is much greater than that of the worst-case. This is because D-Cuts focus on reducing the depth of the decision tree where there is larger traffic. Therefore, if there is no significant traffic flow going though the deepest nodes of the tree, D-Cuts will not reduce the depth of them because it dose not help to reduce the average search time.

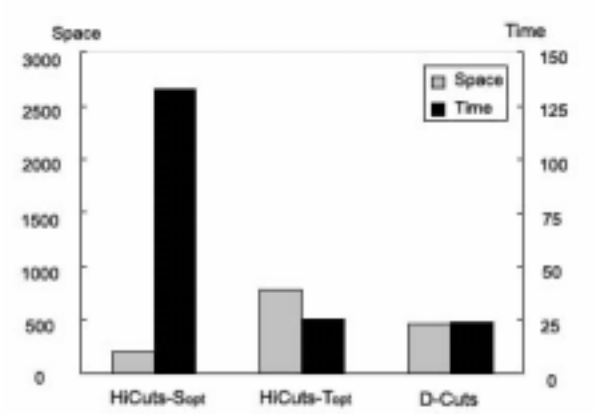

Figure.9 Time/Space trade-offs for SN1. Memory usage (Space) are shown by light histograms, Memory accesses (Time) are shown dark histograms. HiCuts-S_{opt} is space-optimized while HiCuts-T_{opt} is time-optimized.

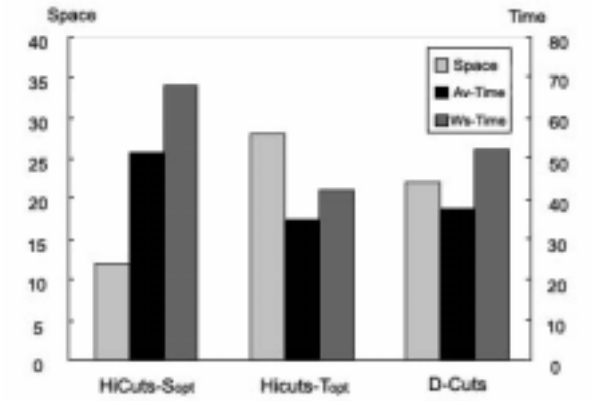

Figure.10 Time/Space trade-offs for FW1. From left to right are memory usage (Space), average memory accesses (Av-Time) and worst-case memory accesses (Ws-Time). HiCuts-S_{opt} is space-optimized while HiCuts-T_{opt} is time-optimized.

Table.3 Memory requirements for the popular packet classification algorithms.

	RFC	ABV	HiCuts (S_{opt})	HiCuts (T_{opt})	D-Cuts
FW1	816	6.2	12	28	23
FW2	910	34.8	15	129	57
CR1	966	1077	85	100	60
CR2	2,220	3,157	2,653	4,235	2,031
SN1	∞ *	2435	202	789	473

*Running Out of Memory

Comparison of memory usage against other popular packet classification algorithms is listed in **Table.3**. The worst-case search time (memory accesses) for the decision tree based algorithms varies from 30 to 130 memory accesses. In comparison, the fastest algorithm

RFC needs just 12 memory accesses while ABV requires up to 200 memory accesses.

G. CONCLUSIONS

Worst-case bounds on search time and memory usage greatly hamper the design of generic algorithms for packet classification. Instead we must search for characteristics of classification problems in pursuit of "fast enough" algorithms using "not too much" memory storage. Our research follows the idea that more heuristic information are adopted, better classification performance are achieved. The presented algorithm D-Cuts (Dynamic Cuttings) bases on both structural characteristics of real-life classifiers (static information) and statistical characteristics of certain network traffic (dynamic information). Experimental result shows that D-Cuts prominently improves the average search time while keeps modest memory usage.

Future work can be conducted to analyze the impact of fast changing traffic patterns and burst caused by virus or other attacks. Future work also includes introducing traffic statistical characteristics into other existing algorithms to develop improved algorithms. The code we wrote for D-Cuts, HiCuts, RFC, and ABV will be publicly available (on line) to encourage experimentation with classification algorithms.

H. ACKNOWLEDGEMENT

The authors would like to express their thanks to Dr. Enke Chen, as well as Quan Huang, for their helps in providing and preparing data for the experiments. Thanks also due to Mr. Dongyi Jiang and other colleagues of RIIT Network Security Lab for their generous suggestions and encouragement.

I. REFERENCE

[1] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Proc. Hot Interconnects*, 1999

[2] M.H. Overmars and A.F. van der Stappen, "Range searching and point location among fat objects," in *Journal of Algorithms, 21(3)*, 1996

[3] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Proc. ACM SIGCOMM 99*, 1999.

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

[5] V. Srinivasan, et al., "Fast and Scalable Layer Four Switching," *Proc. ACM SIGCOMM*, 1998.

[6] F. Baboescu, and G. Varghese, "Scalable Packet Classification," *Proc. ACM SIGCOMM*, 2001.

[7] F. Baboescu, and G. Varghese, "Packet classification Using Multidimensional Cutting," *Proc. ACM SIGCOMM*, 2003.

[8] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?" *Proc. INFOCOM*, 2003.

[9] V.Srinivasan, S.Suri, and G.Varghese, "Packet classification using tuple space search," in *Proc. SIGCOMM*, 1999.

[10] S. Singh and F. Baboescu, "Packet classification repository." http://ial.ucsd.edu/classification

[11] A. Feldman and S. Muthukrishnan, "Tradeoffs for packet classification," in *Proc. INFOCOM*, 2000.

[12] T. Lakshman and D. Stiliadis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proc. SIGCOMM*,1998.