Performance Evaluation and Improvement of Algorithmic Approaches for Packet Classification

Yaxuan Qi, Jun Li
Research Institute of Information Technology (RIIT)
Tsinghua University, Beijing, China, 100084
qiyx98@mails.tsinghua.edu.cn

Abstract

Packet classification is crucial to the implementation of several advanced services that require the capability to distinguish traffic in different flows, such as firewalls, intrusion detection systems and many QoS implementations. Although hardware solutions, such as TCAMs, provide high search rate, they do not scale to large rulesets. Instead, some of the most promising algorithmic research embraces the practice of leveraging the data redundancy in real-life rulesets to improve high performance packet classification. In this paper, we provide a general framework for discerning relationships and distinctions of the design-space of existing packet classification algorithms. We deeply studied several best-known algorithms, such as RFC, HiCuts and HyperCuts according to this framework and suggest for each algorithm an improved scheme. All algorithms we studied, along with their improved version, are objectively accessed using both real-life and synthetic rulesets. The C source codes we wrote for these algorithms are publicly shared on our web-site.[1]

1. Introduction

As the Internet becomes one of the most critical infrastructures of our modern society, keeping network operation and information exchange efficient and secure is highly desired. Traffic engineering, access control, and many other services require a discrimination of packets based on the multiple fields of packet headers, which is called *multidimensional packet classification*.

To reach multi-Gbps packet classification rate, there are currently only a few ASIC/FPGA products. While hardware like Ternary CAMs offers a good solution for small rulesets, they may use too much power and board area for large rulesets. Thus, hardware solutions usually mean higher cost for R&D and production, and lower flexibility in term of modify or upgrade. It is worth looking for alternatives to overcome the limits in hardware solutions, and the challenge of finding efficient algorithmic approaches for packet classification to achieve high performance with comparatively low hardware requirement still motivates the research today.

In this paper, we provide a general framework for discerning relationships and distinctions of the design-space of existing packet classification algorithms. We deeply studied several best-known algorithms, such as *RFC*, *HiCuts* and *HyperCuts*, and suggest for each algorithm an improved scheme. Main contribution of this paper includes:

Dissectional Analysis: From a generic view of space decomposition, we dissect a packet classification problem into several procedures according to the different processes on the search space. The dissectional analysis opens the door for us to develop more efficient packet classification algorithms that leverage on the advantages of other popular algorithms to reach with higher performance.

Novel Ideas: By careful study of the design and implementation of several best-known algorithms, we suggest some novel ideas and present three novel algorithms to further improve the performance of existing schemes.

Objective Evaluations: Thorough comparisons are done with several real-life rulesets, as well as synthetic ones. Experimental results include worst case search time, total memory usage, full update times and performance stability on large rulesets.

The rest of the paper is organized as follows. Section 2 defines the problem of packet classification; Section 3 studies existing algorithms and provides improved schemes; Section 4 illustrates the experimental results; as a summary, Section 5 states our conclusions.

2. Problem Definition

Generic packet classification classifies a packet based on multiple fields of its header. Each rule specifies a *flow* that a packet may belong to, based on

[1] This work is sponsored by the Intel IXA University Program.

certain specifications on the F fields of the packet header. The *flow* uniquely determines the action associated to the rule R, referred to as $R[i]$, is a regular expression on the i^{th} field of the packet header. A packet P is said to *match* a particular rule R, if the i^{th} field of the header of P satisfies the regular expression $R[i]$, for all $0 \le i < F$. If a packet P matches multiple rules, the matching rule with the highest priority is returned.

One possible approach is to map the problem into a geometric point location problem in a multi-dimensional space. It has been proved that the best bounds for point location in N non-overlapping F-dimensional hyper-rectangles are $O(\log^{F-1} N)$ search time with $O(N)$ storage, or $O(\log N)$ search time and $O(N^F)$ storage. In packet classification problem, rules (hyper-rectangles) may overlap, making classification at least as hard as point location (N overlapping rules may yield up to $(2N-1)^F$ non-overlapping hyper-rectangles). Packet classification is made yet more complex by the need to match on ranges as well as prefixes. More specific, A W-bit range can be represented by at most $2(W\text{-}1)$ prefixes, which means a prefix matching algorithm can find ranges with $2W$ times as much storage. Moreover, the large constant hidden in the $O(\cdot)$ notation also impacts actual performance severely in practical implementation.

Although the theoretical bounds make it impossible to design a single algorithm that performs well for *all* cases, fortunately, real-life rulesets have some inherent characteristics that can be exploited to reduce the complexity both in search time and storage space. In literatures [1, 2, 4], a variety of characteristics of real-life rulesets are presented and exploited in proposed algorithms. Some best-known algorithms like *RFC*, *HiCuts* and *HyperCuts* achieved encouraging improvements in performance compared to prior schemes. Their deep thoughts and exhaustive analysis point the way out for further understanding and improvement of multi-dimensional packet classification.

3. Analysis of Existing Algorithms

To unveil the cohering relation lying in different algorithms, we use a *dissectional* methodology to analyze the prior work on packet classification. First we dissect packet classification problem into two generic procedures, then we proceed to the comparison and analysis of existing algorithms according to the different ideas and techniques adopted by them in each procedure.

3.1. Space Decomposition and *Data Structures*

Most existing algorithms adopt a *Divide-and-Conquer* strategy: First divide the original search problem into a series of simplified sub-problems by space *decomposition*; then direct the way of search by building a corresponding *classifier*.[1]

3.1.1. Space Decomposition Schemes

Space decomposition is to partition the search space into certain number of sub-spaces. Each sub-space and the corresponding subset of rules make a new search problem. By recursive decomposition of the search spaces, the complexity of the original classification problem is reduced, so the search result can be obtained by solving a series of sub-problems in stead of doing exhaustive search in the entire search space with all rules. Space decomposition schemes in existing algorithms can be dissected into three main steps: *Segmentation*, *Intersection* and *Aggregation*:

Segmentation: Space segmentation is implemented on a single dimension. The number line of the dimension is divided into segments with certain number of endpoints. There are two schemes in general: one scheme depends on the rule projections (projection-based segmentation) while the other applies equal-sized segmentation.

Intersection: In the segmentation step, the search space is decomposed into sub-spaces along each dimension. Intersection of all these sub-spaces leads to more detailed decomposition on multiple dimensions.

Aggregation: Sub-spaces obtained by segmentation and intersection may have spatial redundancy, e.g. some sub-spaces may contain the same set of rules. Aggregation of such redundant sub-spaces can greatly reduce the storage requirement. There are two ways involved in space aggregation: one is *space combination* for contiguous sub-spaces, and the other is *space mapping* for both contiguous and discontiguous sub-spaces.

3.1.2. Classifier Data Structures

Each packet classification algorithm generates a classifier to direct the way to traverse a series of sub-spaces. More specifically, the classifier determines how to locate a point into its corresponding sub-space and how to go from the current search space to the next. There are two types of Data Structures adopted by different classifiers: One is pointer-based *decision trees* and the other is index-based *lookup tables*.

[1] Classifier in this paper refers to the whole data-structure generated by algorithms for packet search. However, in some literature, classifier may refer to a set of rules, i.e. the ruleset defined in this paper.

Decision Trees: Pointer-based decision trees algorithms partition the search space into 2^w equal-sized sub-spaces at each space decomposition stage, where *w* is called the *stride*. Each internal node contains a corresponding search space, a set of rules, the information for packet search, as well as pointers to child nods. The final search results (identifiers of the best matching rules) are saved in leaf nodes.

Lookup Tables: All the entries of an index-based lookup table are stored in consecutive memories. The indices of the table are obtained by space mapping. Each entry corresponds to a particular sub-space and stores the search result at current stage.

According to the space decomposition techniques and classifier data structures, the following part of this section will go deep into the analysis of three packet classification algorithms, including *RFC*, *HiCuts* and *HyperCuts,* which achieve the best-reported performances in existing literatures.

3.2. Recursive Flow Classification (*RFC*)

Gupta and McKeown introduced a multi-dimensional algorithm named *Recursive Flow Classification (RFC)*, which provides high lookup rates at the cost of memory inefficiency [2]. The authors performed a rather comprehensive and wildly cited study of real-life rulesets and extracted several useful characteristics. Specifically, they noted that rule overlap is much smaller than the worst-case of $O(N^F)$. *RFC* attempts to recursively map an *S*-bit packet header to a *T*-bit action identifier, where $T \ll S$. At each stage the algorithm maps one set of values to a smaller set, and in each phase a set of memories return a value shorter than the index of the memory access.

3.2.1. Space Decomposition Scheme

RFC performs independent, parallel searches on *chunks* of the packet header, the result of the chunk searches are combined in multiple *phases*. In the first phase, *F* fields of the packet header are **segmented** according to unique rule-projection intervals into multiple chunks (sub-spaces) that are used to index into multiple memories. Sub-spaces associated with same rules will be labeled with same eqID and then **aggregated**. In subsequent phases, earlier sub-spaces obtained from one dimensional segmentation are recursively **intersected** with the sub-spaces obtained from other dimensions. In the final phase, the memory yields the action.

3.2.2. Classifier Data Structure

RFC searches in chunk and aggregation utilizes index-based **lookup tables**: the address for the table lookup is formed by concatenating the eqIDs from the previous stages. The resulting eqID is smaller than the address; thus *RFC* performs a multi-stage reduction to a final eqID that specifies the action to apply to the packet.

3.2.3. Evaluation and Improvement

The use of indexing simplifies the lookup process at each stage and allows *RFC* to provide very high throughput. Because searches in a indexed table needs only one memory access, *RFC* achieves $O(F)$ search rate (F is on the same order of the number of lookup tables). However, this simplicity and performance comes at the cost of memory inefficiency.

In the first phase, *RFC* uses the number line (all possible values in a single dimension) as the indices of the lookup tables in pursuit of $O(1)$ search rate. Although such a scheme is feasible for 2^{16} port numbers and 2^8 protocol fields, it is impractical for 2^{32} (for IPv4) IP addresses. Gupta suggested splitting the 32-bit IP address into two 2^{16}-entry independent chunks. Such a splitting works well for fast search but increases the number of intermediate sub-spaces because a single rule may appear twice in the two 2^{16}-entry chunks. Moreover, the coming 128-bits IP address will make it more unfeasible to apply *RFC* to IPv6 networks.

To avoid the excessive number of indices, we implement binary searches rather than table lookups on source/destination IP fields in the first phase. Because *N* rules lead to at most 2*N*-1 segments in each dimension, a binary search to locate a packet in its corresponding sub-space can be performed in $O(\log N)$ time. Therefore, the binary search scheme is independent on the range of IP addresses, and hence does not require the huge lookup tables for IP address even for IPv6. This idea is adopted by an improved version of *RFC* and has been published in one of our technical papers [3]. Experimental results in next section show that this approach uses 2 to 20 times less memory than *RFC*, and remains a relatively fast search rate.

3.3. *HiCuts* and *HyperCuts*

HiCuts [1] and its improved version *HyperCuts* [4] are seminal techniques provide best time/space tradeoffs in existing literatures. *HiCuts* preprocesses the rulesets in order to build a decision tree with leaves containing a small number of rules bounded by a threshold (*binth* in [1]). Packet header fields are used to traverse the decision tree until a leaf is reached. The rules stored in that leaf are then linearly searched for a match. *HyperCuts* improves upon the *HiCuts* algorithm by applying multi-dimensional space decomposition at each internal node.

3.3.1. Space Decomposition Scheme

HiCuts decomposes the multi-dimensional search space guided by heuristics that exploit the characteristic of real-life rulesets. At each internal node, the current search space is cut (**segmented)** into certain number of equal-sized sub-spaces along a particular dimension. The number of cuttings and the dimension to cut is determined by heuristics (see [1] for the heuristics used). Different from *HiCuts*, *HyperCuts* performs segmentation on multiple fields at each internal node. The number of cuttings and the fields to cut also selected by heuristics (see [4]). The sub-spaces obtained on each fields are **intersected** and each intersection generates a child node. Both *HiCuts* and *HyperCuts* **aggregate** contiguous sub-spaces if they share the same set of rules.

3.3.2. Classifier Data-structure

HiCuts and *HyperCuts* build **decision trees** with leaves containing a small list of rules as the classifier's data-structure. Each node of the tree represents the current search space. The root node represents the entire search space, which is partitioned into smaller sub-spaces, represented by is child nodes. Each sub-space is recursively partitioned until no sub-space has more than *binth* rules, where *binth* is a tunable parameter.

To link the current node with its children, *HiCuts* stores a pointer array at each node. Each pointer in the array corresponds to a sub-space and sequentially stored according to the order of the sub-space. Due to space aggregation, consecutive pointers may point to a single child node. *HyperCuts*, however, encodes sub-spaces using pointer matrices, which allows the data-structure to make multiple cuts in multiple dimensions.

3.3.3. Evaluation and Improvement

The first advantage of *HiCuts* and *HyperCuts* is the hierarchical space decomposition scheme. In each classification stage, the decision tree just examines w of the S bits of the packet header, where w is the various stride specifies the number of cuttings. The point location in an internal node is virtually implemented within a w-bit degenerate space rather than the entire search space.

The second advantage comes from the heuristics they used in building the decision tree. By exploring the characteristics of rulesets, *HiCuts* and *HyperCuts* make "intelligent cuttings" that significantly reduce the spatial redundancy in corresponding rules.

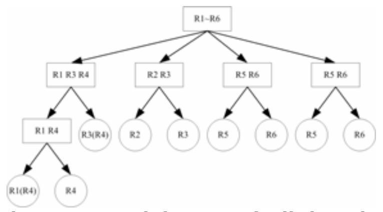

Figure 1. Decision tree built by HiCuts
The decision tree build by *HiCuts* using the sample ruleset has 15 tree nodes (6 internal nodes and 9 leaf nodes). The depth of the decision tree is 4.

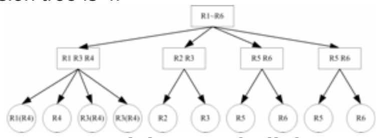

Figure 2. Decision tree built by D-Cuts
The decision tree build by *D-Cuts* using the sample ruleset has 15 tree nodes (5 internal nodes and 10 leaf nodes). The depth of the decision tree is 3. Compared to *HiCuts*, *D-Cuts* improves the search rate by cutting down the (local) depth of the decision tree without a significant increase of memory usage.

Finally, the hybrid-data structure effectively cuts down the storage requirement. Different from the *complete* de-overlapping process in *RFC*, decision trees in *HiCuts* and *HyperCuts* perform *incomplete* space decomposition, i.e. the sub-spaces in leaf nodes contain more than one possible matching rule. Because the de-overlapping even on a small number of rules may lead to large number of space partitions ($O(N^F)$), the linear search on the final rule-lists greatly reduce the spatial complexity.

Although *HiCuts* and, especially *HyperCuts*, are superior to most existing algorithms, they still have some inherent disadvantages and can be further improved. First, they only exploit the *static* characteristics in the real-life rulesets while assume all incoming packets are distributed uniformly in the search space. However, it is unlikely that the traffic in a certain network uniformly spread over all IP addresses and/or port numbers. Each network has its own traffic patterns, and the packet classification process is affected by the *dynamic* characteristics to a certain extend. We improved the original *HiCuts* by introducing dynamic characteristics of network traffic in building the decision tree. The proposed algorithm *D-Cuts* [6] elegantly refines the space allocation function in *HiCuts* by introducing traffic statistics, which makes the number of cuttings not only in proportion to the number of rules, but also to the volume of traffic that "flows" through the current search space. Figure 1 and Figure 2 depict the different decision trees built by *HiCuts* and *D-Cuts* respectively.

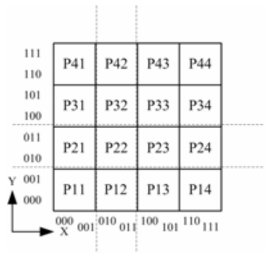

Figure 3. Pointer Matrix
The 4x4 pointer matrix maps each of the 16 sub-spaces into corresponding child nodes.

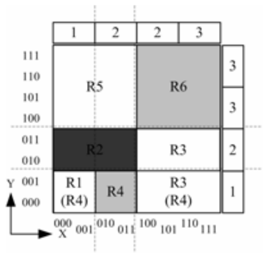

Figure 4. ID Index
The two 4-entry Space ID arrays replaces the 4x4 pointer matrix.

Another disadvantage lies in the pointer arrays, especially the pointer matrices in *HyperCuts*. Each internal node has a pointer matrix that stores 2^{fw} pointers, where f is the number of cutting dimensions and w is the stride. To limit the size of such a pointer matrix, *HyperCuts* bounds the stride by space allocation function. However, smaller strides in internal nodes tend to increase the depth of the decision tree, and hence result in a slower search rate. In our research, we significantly reduce the memory requirement by replacing the 2^{fw} pointers with $f * 2^{w}$ space IDs. The idea of this technique is briefly shown in Figure 3 and Figure 4. More details can be found in another technical paper [9].

4. Experimental Results

The algorithms in the comparison experiments include *RFC*, *HSM*, *HiCuts*, *D-Cuts*, *HyperCuts* and *sBits*. We make our best effort to make sure the fairness of our result analysis. Experimental results show that our codes achieved nearly the same performance compared to the results reported in [4].

4.1. Rulesets

Evaluations are done on real-life firewall and core router rulesets obtained from enterprise networks and major ISPs. Firewall rulesets are named FW1, FW2, FW3 and Core router rulesets are named CR1, CR2, CR3, CR4. All rules are 5-dimensional with 32-bit source/destination IP addresses represented as prefixes, 16-bit source/destination port numbers represented as ranges and an 8-bit protocol.

4.2. Metrics

All the algorithms in our experiment are written in C codes and running in a PC with Pentium4 2.4GHz CUP. We examine, for each ruleset, the number of *memory accesses* (Time) and the amount of *memory usage* (Space). Different from [4] (where one memory access is a single 32-bit word access) one memory access here refers to reading a certain number (1~8) of continuous memory words. This is because today's most on-chip SRAM support burst mode reading, i.e. the time spent in reading continuous memory is very close to that of reading a single word.

4.3. Performance Comparison

Table 1 shows the memory comparison of *RFC* and *HSM* on real-life rulesets. We can see from the table that, for all firewall rulesets, *HSM* achieves outstanding performance, using approximately 5~20 times less memory than *RFC*. For the larger core router rulesets, *HSM* is still superior to *RFC*, using 40% ~ 60% less memories than *RFC*.

Search speed of both *RFC* and *HSM* are on the same order. It is reported in [2] that *RFC* uses 12 memory accesses for 4-field core router sets. In comparison, *HSM* uses 15~25 memory access for all our rulesets, and in the worst-case, *HSM* uses less than 30 memory accesses for 4-field rulesets with up to 4000 rules.

Memory comparison between *HiCuts* and *D-Cuts* is shown in Table 2. *D-Cuts* uses about 50% memory of that of *Hi-Cuts* while keeps the same search rate (even faster in the average-case). Although the performance of *D-Cuts* depends on the stability of network traffics and the accuracy of the sampling statistics, in our study we might assume that these dynamic characteristics are "good enough" as well, because network statistic and modeling are other topics in network research.

Table 3 is the memory usage comparison of *HiCuts*, *HyperCuts* and *sBits*. Compared to the best-reported algorithm *HyperCuts*, *sBits* uses about 20~80 times less memory than *HyperCuts*. This outstanding performance results from the matrix-to-index conversion, which significantly reduces the memory usage caused by the redundant pointer matrices. Because *sBits* chooses a relatively large stride, it also has superior performance in search speed (see Table 4).

More heuristics are used in algorithms, more preprocessing time is needed. Algorithms like *RFC* and *HiCuts* both consume a lot of preprocessing time in building the classifiers. *sBits* significantly reduces the preprocessing time by simplification of the heuristics used in space aggregation and segmentations (see [9] for more details). Table 5 gives the preprocessing time for *sBits*, in comparison with *RFC* and *HiCuts*.

Table 1. Memory usage comparison
RFC & *HSM* (Unit: 32-bit word)

	No. Rules	*RFC*	HSM
FW1	68	200,652	10,223
FW2	136	209,602	27,657
FW3	340	296,382	65,581
CR1	500	264,987	29,814
CR2	1000	530,539	230,716
CR3	1530	863,476	486,857
CR4	1945	1,580,005	989,161

Table 2. Memory usage comparison
HiCuts & *D-Cuts* (Unit: 32-bit word)

	No. Rules	*HiCuts*	*D-Cuts*
FW2	136	10,779	5,752
FW3	340	24,645	16,231
CR1	500	29,409	15,163
CR2	1000	979,736	507,930

Table 3. Memory usage comparison
HiCuts, HyperCuts and sBits (Unit: 32-bit word)

	No. Rules	*HiCuts*	*HyperCuts*	sBits
FW1	68	5,443	35,401	420
FW2	136	10,779	69,782	924
FW3	340	24,645	172,932	2,331
CR1	500	29,409	89,005	3,612
CR2	1000	979,736	871,541	28,287
CR3	1530	13,606,858*	480,225	29,204
CR4	1945	5,928,724*	672,442	43,183

Table 4. Worst case search time comparison
sBits vs. *HiCuts*/*HyperCuts*. (Unit: Memory Access)

	No. Rules	*HiCuts*	*HyperCuts*	*sBits*
FW1	68	19	16	15
FW2	136	20	16	15
FW3	340	20	16	15
CR1	500	24	17	16
CR2	1000	30	17	16
CR3	1530	36	18	16
CR4	1945	34	18	17

Table 5. Preprocessing time comparison
sBits vs. *RFC* & *HiCuts* (Unit: milliseconds)

	No. Rules	*RFC*	*HiCuts*	*sBits*
FW1	68	1,492	73	1
FW2	136	1,762	211	22
FW3	340	3,185	465	40
CR1	500	4,597	409	56
CR2	1000	12,929	7661	261
CR3	1530	37,754	22,522	281
CR4	1945	67,087	21,248	350

5. Conclusion

In this paper, we first make a dissectional analysis for existing algorithms to find their cohering relations, and then proposed improved schemes for three best-known algorithms, including *RFC*, *HiCuts* and *HyperCuts*. Experimental results show that our schemes outperform the best results of the original algorithms. We use the incisive conclusion by Gupta in [1] as a summary for the development of packet classification algorithms: “The theoretical bounds tell us that it is not possible to arrive at a practical worst case solution. Fortunately, we don't have to; No single algorithm will perform well for all cases. Hence a hybrid scheme might be able to combine the advantages of several different approaches.”

Future work can be conducted to introduce network traffic statistics into other packet classification algorithms. Future work also includes the implementation of the proposed algorithms on new generation network processors. The codes we wrote for *RFC*, *HSM*, *HiCuts*, *D-Cuts*, *HyperCuts* and *sBits* will be publicly available to encourage experimentation with classification algorithms.

6. References

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

[2] P. Gupta and N. McKeown, “Packet classification on multiple fields,” Proc. ACM SIGCOMM 99, 1999.

[3] Bo Xu, Dongyi Jiang, Jun Li, “HSM: A Fast Packet Classification Algorithm”, The IEEE 19th International Conference on Advanced Information Networking and Applications (AINA), Taiwan, 2005.

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

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

[6] Yaxuan Qi, Jun Li, “Dynamic Cuttings: Packet Classification with Network Traffic Statistics”, The 3rd International Trusted Internet Workshop (TIW), India, 2004.

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

[8] David E. Taylor “Survey & Taxonomy of Packet Classification Techniques”, Washington University in Saint-Louis, US, 2004.

[9] Yaxuan Qi and Jun Li, “Multidimensional Packet Classification with Shifting Bits”, Submitted to Proc. of 14th International conference on computer communications and networks (ICCCN), San Diego, California USA, October 17-19, 2005.